TC2100 PK2

P.O. BOX 1450
ALEXANDRIA, VA 22313-1450
IF UNDELIVERABLE RETURN IN TEN DAYS

OFFICIAL BUSINESS

AN EQUAL OPPORTUNITY EMPLOYER

RETURNED TO SENDER
ATTEMPTED — NOT KNOWN

FEB 202

1 of 2

[illegible]

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/812,542	03/20/2001	Kiyotaka Okawa	8279.310US01	2559

7590 08/12/2004

Merchant, Gould, Smith, Edell,
Welter & Schmidt
3100 Norwest Center
90 South Seventh Street
Minneapolis, MN 55402-4131

EXAMINER	
KLINGER, SCOTT M	
ART UNIT	**PAPER NUMBER**
2153	

DATE MAILED: 08/12/2004

3

Please find below and/or attached an Office communication concerning this application or proceeding.

RECEIVED
AUG 2 3 2004
Technology Center 2100

PTO-90C (Rev. 10/03)

[illegible]

Office Action Summary	Application No.	Applicant(s)	
	09/812,542	OKAWA ET AL.	
	Examiner	Art Unit	
	Scott M. Klinger	2153	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 20 March 2001.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-5 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-5 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

Claims 1-5 are pending.

Priority

A claim for foreign priority has been made. Receipt is acknowledged of papers submitted under 35 U.S.C. 119(a)-(d), which papers have been placed of record in the file. The effective filing date for subject matter in the application is 22 March 2000.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 1-5 are rejected under 35 U.S.C. 103(a) as being unpatentable over Thackston (U.S. Patent Number 6,295,513, hereinafter "Thackston") in view of Saucedo et al. (U.S. Patent Number 5,754,738, hereinafter "Saucedo").

In referring to claim 1, Thackston discloses a network-based system for the manufacture of parts with a virtual collaborative environment for design, development, and fabricator selection. Thackston shows substantial features of the claimed invention, including:

- Figure data storage means for storing figure data on a whole including an object of retrieval and for transmitting the figure data to the client in response to an instruction from the client:

  "It is another object of the present invention to provide such a network-based system whereby a central server maintains engineering data, such as design

documents and three dimensional model data, in a common, neutral format, which is accessible by authorized team members through a graphical user interface that is substantially platform independent to reduce or eliminate the necessity for specialized hardware and software." (Thackston, col. 3, line 64 – col. 4, line 4)

- Parts information storage means for storing parts information:
  Thackston, Fig. 2 shows database 210, where the parts information is stored
- Figure information storage means for storing figure information including coordinate data with respect to the figure data:
  Thackston, Fig. 2 shows database 210, where the parts information is stored; a system that stores CAD model data inherently implies coordinate data
- The client comprises a general-purpose web browser for showing the parts information and the figure data:
  "Prime contractor user systems 220 may comprise computers running standard operating systems and supporting "browser" technologies for accessing and displaying data over a common network, such as personal computers with Windows NT™ and Microsoft Internet Explorer ™ 5.0 or Netscape Communicator ™ 4.06, an Apple Macintosh™ running MOSAIC™ web browser software, a Sun SPARCstation ™ running UNIX and Netscape Communicator ™, and a Silicon Graphics ™ UNIX-based workstation such as the SGI Octane ™ running Netscape Communicator™." (Thackston, col. 9, lines 53-62)

However, Thackston does not discuss the display of the parts in great detail. Thackston does not explicitly show linking the parts information and the figure information to each other bidirectionally or showing the parts list and figure information simultaneously. Nonetheless this feature is well known in the art and would have been an obvious implementation of the system disclosed by Thackston as evidenced by Saucedo.

In analogous art, Saucedo discloses a computerized prototyping system employing virtual system design environment. Saucedo shows linking the parts information and the figure information to each other bidirectionally or showing the parts list and figure information simultaneously: Saucedo, Fig. 25 is an illustration shown the example of an interface of the design browser; The model, component files (parts), and a hierarchical graph (of how the parts are linked) are displayed simultaneously.

Given these teachings, a person of ordinary skill in the art would have readily recognized the desirability and advantages of implementing the system of Thackston so as to link the parts information and the figure information to each other bidirectionally and show the parts list with the figure information simultaneously, such as taught by Saucedo, in order to provide a view of the overall system while a specific part is edited.

In referring to claim 2, Thackston in view of Saucedo shows,

- The mark-up language is Extensible Mark-up Language (XML):
  "In one embodiment, prime contractor user system 220 comprises a personal computer or workstation running a standard operating system such as Windows NT, and using a standard browser such as Microsoft Internet Explorer ™ 5.0 capable of interpreting HTML 4.0, XML, VRML, and running Java ™ applets" (Thackston, col. 10, line 5-10)

In referring to claim 3, Thackston in view of Saucedo shows,

- The figure data are image data, which do not have attribute of coordinates:
  Thackston, Fig. **12** shows that some of the figure data can comprise drawing symbols **1206**, which do not have coordinate data

In referring to claim 4, Thackston in view of Saucedo shows,

- When a retriever selects one of the objects of retrieval which are shown on the general-purpose web browser, the one changes visually on the general-purpose web browser:
  "The graphics capability allows a prime contractor and prospective fabricator to view the three-dimensional part design, including the execution of various manipulations, such as virtual rotations and translations, pan, zoom and 'fly throughs.'" (Thackston, col. 5, line 64 – col. 6, line 1)

In referring to claim 5, Thackston in view of Saucedo shows,

- External output means for allowing a retriever to take out a result of the retrieval and to make use of the result:
  "Stored standard contracts data module 696 may comprise a series of contract "templates" for prime contractors and suppliers to use as a starting point for creating an agreement. For example, in one embodiment there is a standard form agreement for a fabricator to produce a quantity of prototypes of a design within some time-frame." (Thackston, col. 13, line 11-16)

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Scott M. Klinger whose telephone number is (703) 305-8285. The examiner can normally be reached on M-F 7:00am - 3:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Glenn Burgess can be reached on (703) 305-4792. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Scott M. Klinger
Examiner
Art Unit 2153

smk

[illegible]

GLENTON B. BURGESS
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

Notice of References Cited	Application/Control No. 09/812,542	Applicant(s)/Patent Under Reexamination OKAWA ET AL.	
	Examiner Scott M. Klinger	Art Unit 2153	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-5,201,046	04-1993	Goldberg et al.	707/100
	B	US-5,720,023	02-1998	Putland et al.	345/440
	C	US-5,754,738	05-1998	Saucedo et al.	706/11
	D	US-5,761,063	06-1998	Jannette et al.	700/97
	E	US-5,761,674	06-1998	Ito, Kenji	707/104.1
	F	US-6,067,091	05-2000	Itagaki et al.	345/629
	G	US-6,295,513	09-2001	Thackston, James D.	703/1
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

US005201046A

United States Patent [19]

Goldberg et al.

[11] Patent Number: 5,201,046

[45] Date of Patent: Apr. 6, 1993

[54] RELATIONAL DATABASE MANAGEMENT SYSTEM AND METHOD FOR STORING, RETRIEVING AND MODIFYING DIRECTED GRAPH DATA STRUCTURES

[75] Inventors: Robert N. Goldberg, Redwood City; Gregory A. Jirak, La Honda, both of Calif.

[73] Assignee: Xidak, Inc., Palo Alto, Calif.

[21] Appl. No.: 542,163

[22] Filed: Jun. 22, 1990

[51] Int. Cl.5 .... G06F 15/419

[52] U.S. Cl. 395/600; 364/DIG. 1; 364/282.1; 364/283.4

[58] Field of Search 364/200, 300; 395/600

[56] References Cited

U.S. PATENT DOCUMENTS

4,468,732	8/1984	Raver	364/200
4,829,427	5/1989	Green	364/300
4,918,593	4/1990	Huber	364/200

OTHER PUBLICATIONS

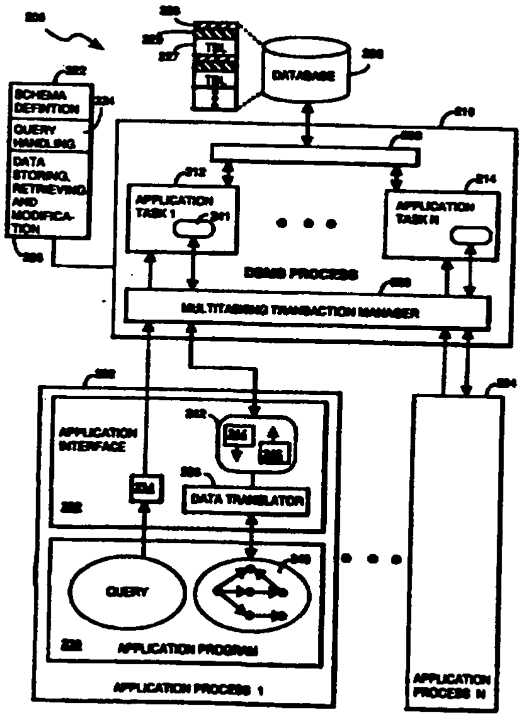
McFadden et al; "Data Base Management", 1985, pp. 122-150, The Benjamin/Cummings Publishing Company, Menlo Park, CA.

Primary Examiner—David L. Clark
Assistant Examiner—Wayne Amsbury
Attorney, Agent, or Firm—Flehr, Hohbach, Test, Albritton & Herbert

[57] ABSTRACT

An improved database management system (DBMS) stores, retrieves and manipulates directed graph data structures in a relational database. Each directed graph data structure contains one or more records of data which are interconnected by pointers. Data is stored in the database in the form of two dimensional tables, also known as flat files. The improved DBMS defines a schema for each table in the database. The schema defines the name and data type of each column in a database table. In tables used to store directed graph data structures, at least one column will be defined as having a reference data type. Non-empty entries in that column are pointers to rows in a specified table. Directed graph data structures are stored in specified tables by storing each record of the directed graph in a distinct row of one of the specified tables, with references corresponding to interconnections between records being stored in reference data type columns. Portions of a directed graph are retrieved from the specified table, in accordance with a single specified query and then the query is automatically expanded by also retrieving additional portions of the tables which are referenced by the previously retrieved portions, thereby performing a transitive closure. The retrieved data is stored in a buffer as a list of rows, and then communicated to an application process. An interface program converts the list of rows stored in the buffer into a directed graph data structure.

15 Claims, 6 Drawing Sheets



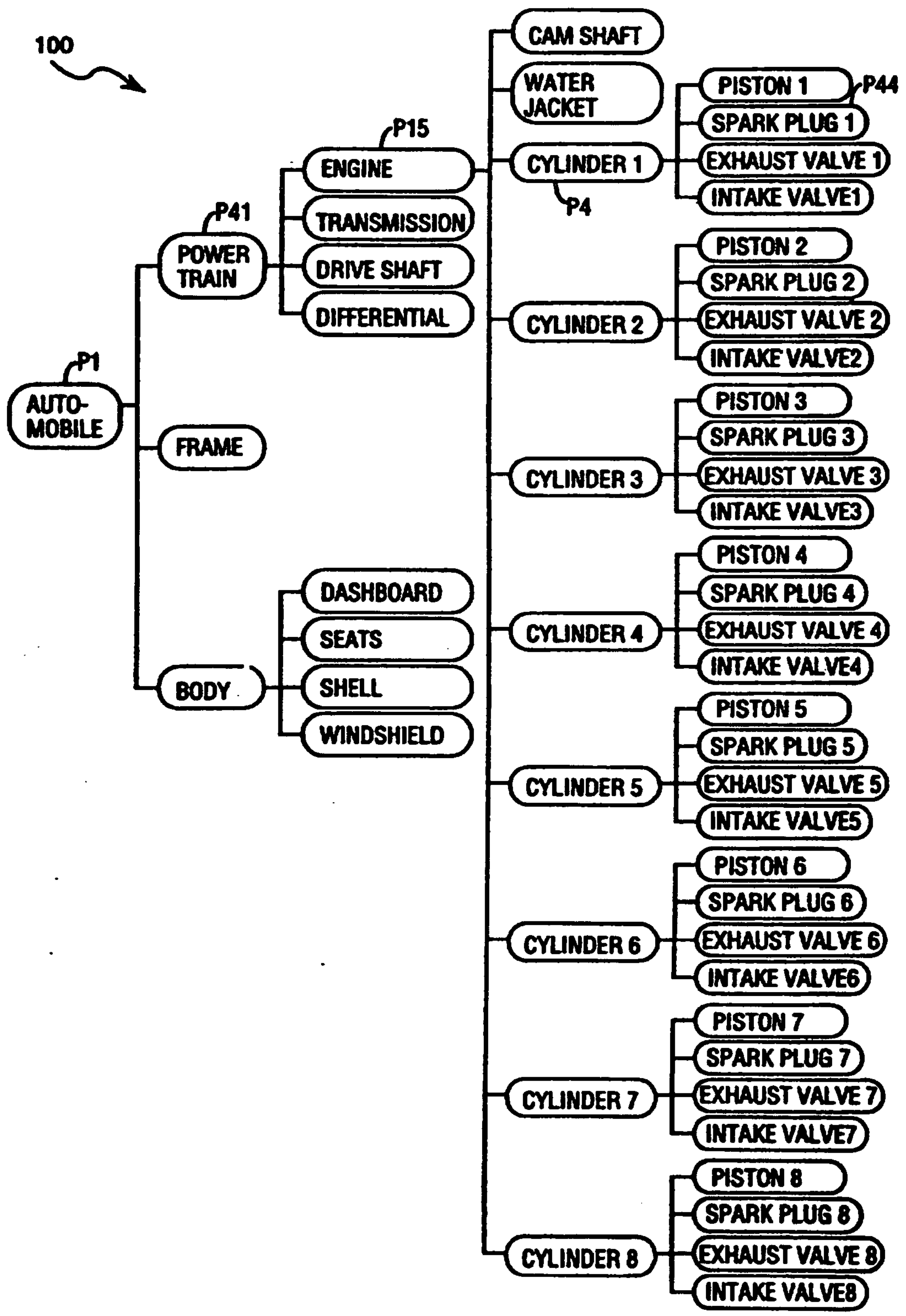

100
P1
AUTO-MOBILE
P41
POWER TRAIN
FRAME
BODY
P15
ENGINE
TRANSMISSION
DRIVE SHAFT
DIFFERENTIAL
DASHBOARD
SEATS
SHELL
WINDSHIELD
CAM SHAFT
WATER JACKET
CYLINDER 1
P4
CYLINDER 2
CYLINDER 3
CYLINDER 4
CYLINDER 5
CYLINDER 6
CYLINDER 7
CYLINDER 8
PISTON 1
P44
SPARK PLUG 1
EXHAUST VALVE 1
INTAKE VALVE1
PISTON 2
SPARK PLUG 2
EXHAUST VALVE 2
INTAKE VALVE2
PISTON 3
SPARK PLUG 3
EXHAUST VALVE 3
INTAKE VALVE3
PISTON 4
SPARK PLUG 4
EXHAUST VALVE 4
INTAKE VALVE4
PISTON 5
SPARK PLUG 5
EXHAUST VALVE 5
INTAKE VALVE5
PISTON 6
SPARK PLUG 6
EXHAUST VALVE 6
INTAKE VALVE6
PISTON 7
SPARK PLUG 7
EXHAUST VALVE 7
INTAKE VALVE7
PISTON 8
SPARK PLUG 8
EXHAUST VALVE 8
INTAKE VALVE8

FIGURE 1

110

PartName	containsPartName
AUTOMOBILE	BODY
AUTOMOBILE	FRAME
AUTOMOBILE	POWER TRAIN
BODY	DASHBOARD
BODY	SEATS
BODY	SHELL
BODY	WINDSHIELD
CYLINDER 1	EXHAUST VALVE 1
CYLINDER 1	INTAKE VALVE1
CYLINDER 1	PISTON 1
CYLINDER 1	SPARK PLUG 1
CYLINDER 2	EXHAUST VALVE 2
CYLINDER 2	INTAKE VALVE 2
CYLINDER 2	PISTON 2
CYLINDER 2	SPARK PLUG 2
. . . .	. . . .
CYLINDER 8	EXHAUST VALVE 8
CYLINDER 8	INTAKE VALVE8
CYLINDER 8	PISTON 8
CYLINDER 8	SPARK PLUG 8
ENGINE	CAM SHAFT
ENGINE	CYLINDER 1
ENGINE	CYLINDER 2
ENGINE	CYLINDER 3
ENGINE	CYLINDER 4
ENGINE	CYLINDER 5
ENGINE	CYLINDER 6
ENGINE	CYLINDER 7
ENGINE	CYLINDER 8
ENGINE	WATER JACKET
POWER TRAIN	DIFFERENTIAL
POWER TRAIN	DRIVE SHAFT
POWER TRAIN	ENGINE
POWER TRAIN	TRANSMISSION

120

PartName	Cost
AUTOMOBILE	15,000.00
BODY	8,000.00
CAM SHAFT	500.00
CYLINDER 1	250.00
CYLINDER 2	250.00
. . . .	. . . .
CYLINDER 7	250.00
CYLINDER 8	250.00
DASHBOARD	2,000.00
DIFFERENTIAL	500.00
DRIVE SHAFT	500.00
EXHAUST VALVE 1	47.50
EXHAUST VALVE 2	47.50
. . . .	. . . .
EXHAUST VALVE 8	47.50
FRAME	2,000.00
INTAKE VALVE 1	47.50
INTAKE VALVE 2	47.50
. . . .	. . . .
INTAKE VALVE 8	47.50
PISTON 1	150.00
PISTON 2	150.00
. . . .	. . . .
PISTON 8	150.00
POWER TRAIN	5,000.00
SEATS	1,000.00
SHELL	4,500.00
SPARK PLUG 1	5.00
SPARK PLUG 2	5.00
. . . .	. . . .
SPARK PLUG 8	5.00
TRANSMISSION	1,000.00
WATER JACKET	500.00
WINDSHIELD	500.00

FIGURE 2

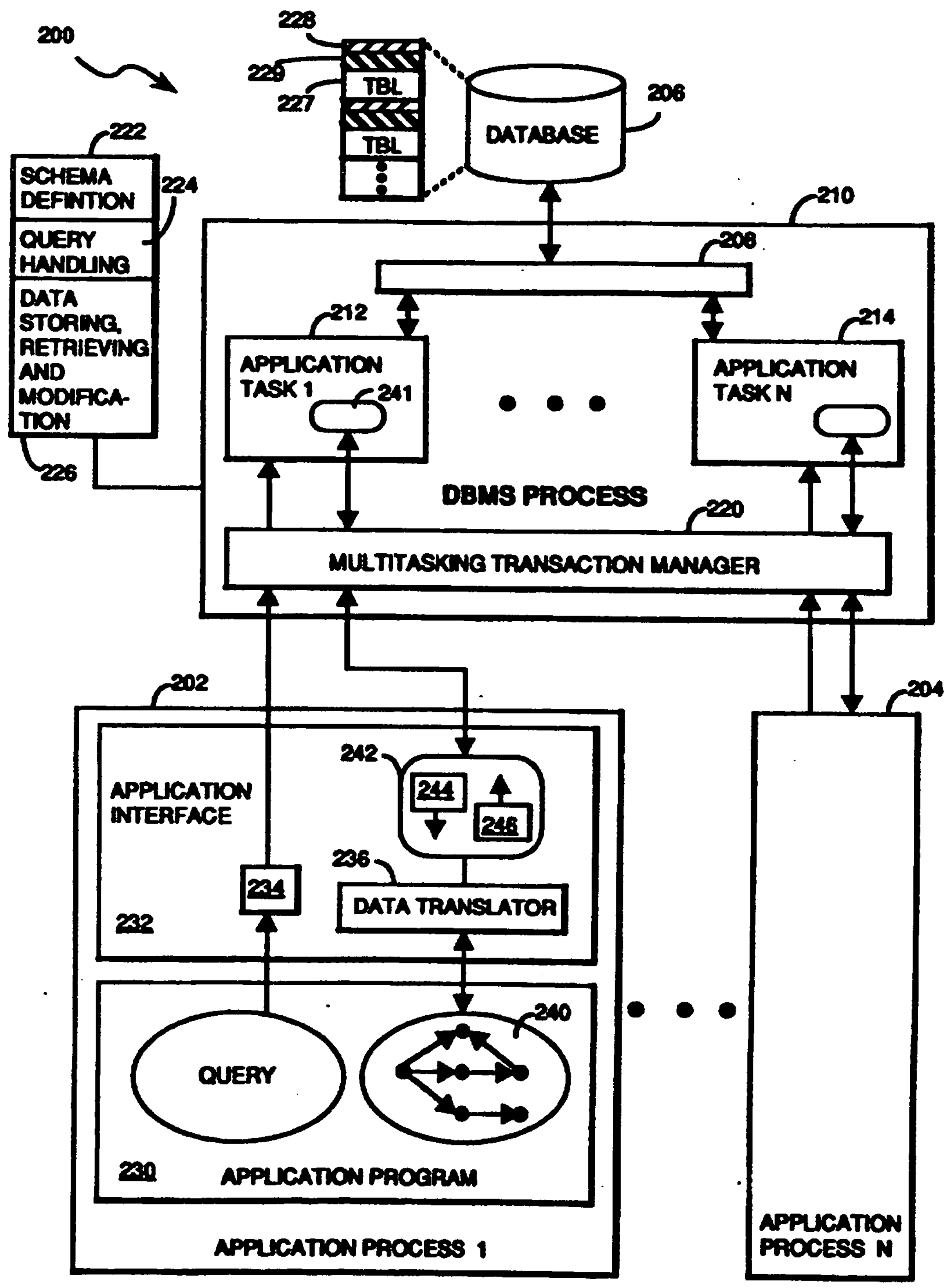

200
228
229
227
TBL
TBL
DATABASE
206
222
SCHEMA DEFINTION
224
QUERY HANDLING
DATA STORING, RETRIEVING AND MODIFICATION
226
210
208
212
APPLICATION TASK 1
241
214
APPLICATION TASK N
DBMS PROCESS
220
MULTITASKING TRANSACTION MANAGER
202
204
APPLICATION INTERFACE
242
244
246
236
234
DATA TRANSLATOR
232
240
QUERY
230
APPLICATION PROGRAM
APPLICATION PROCESS 1
APPLICATION PROCESS N

FIGURE 3

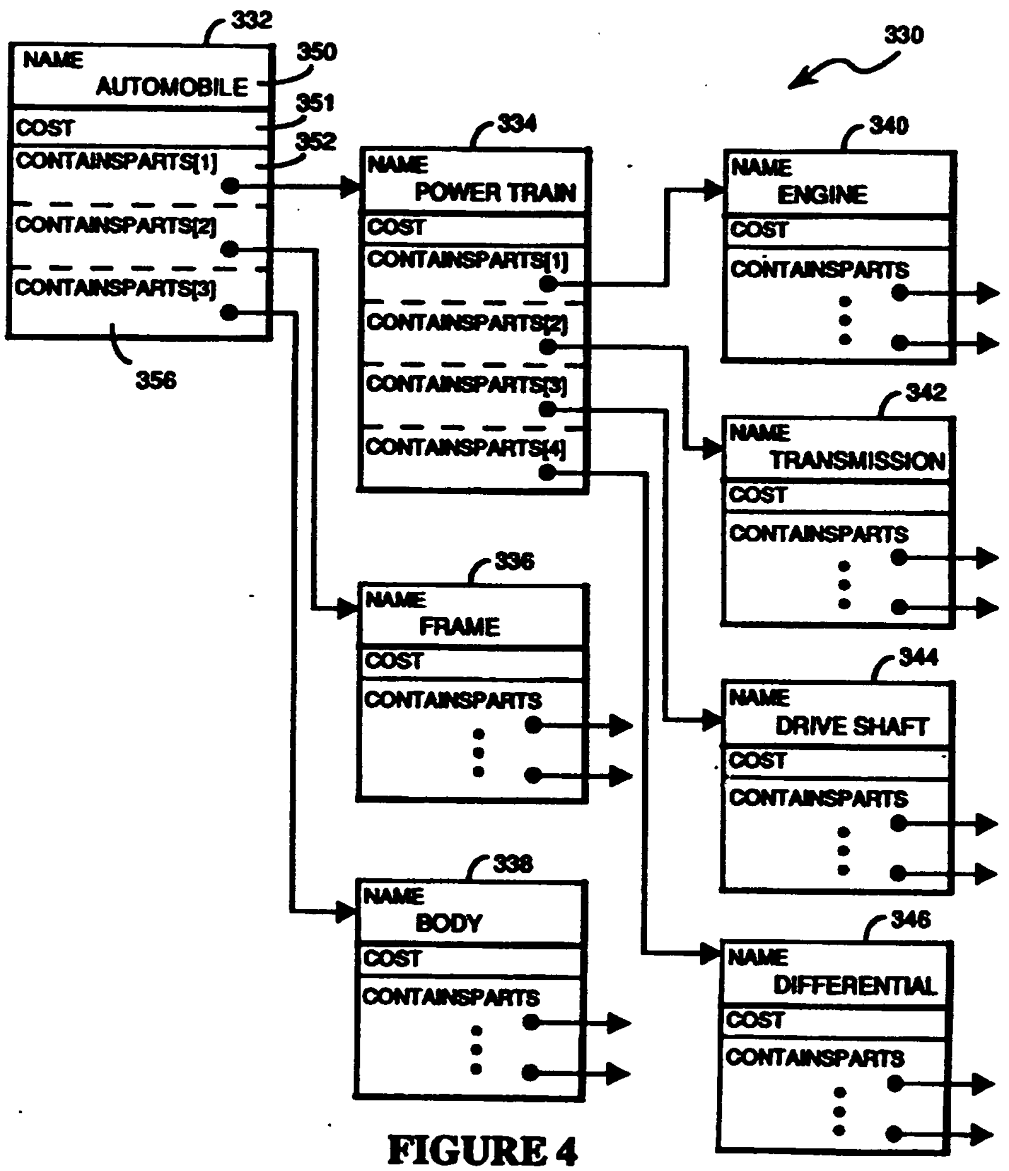
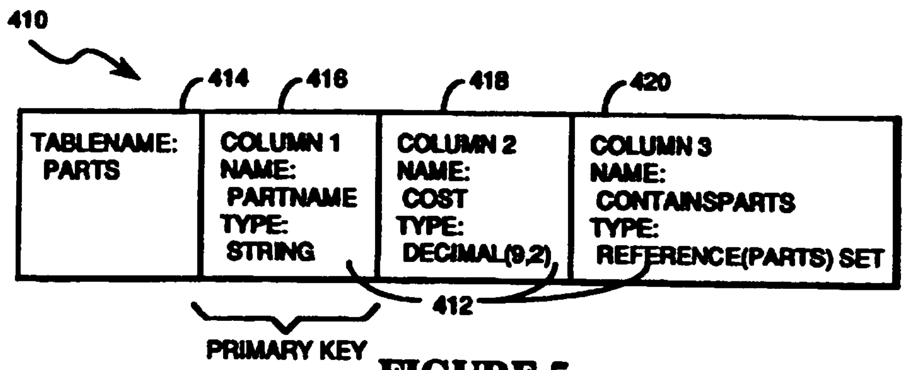


FIGURE 4



410
414
416
418
420
TABLENAME: PARTS
COLUMN 1 NAME: PARTNAME TYPE: STRING
COLUMN 2 NAME: COST TYPE: DECIMAL(9,2)
COLUMN 3 NAME: CONTAINSPARTS TYPE: REFERENCE(PARTS) SET
412
PRIMARY KEY

FIGURE 5

400 402 404 406

PartName	COST	CONTAINSPARTS
AUTOMOBILE	15,000.00	POWER TRAIN FRAME
BODY	8,000.00	DASHBOARD SEATS SHELL
CAM SHAFT	500.00	
CYLINDER 1	250.00	PISTON 1 SPARK PLUG 1 EXHAUST VALVE 1
CYLINDER 2	250.00	PISTON 2 SPARK PLUG 2 EXHAUST VALVE 2
• • • •	• • • •	
CYLINDER 7	250.00	PISTON 7 SPARK PLUG 7 EXHAUST VALVE 7
CYLINDER 8	250.00	PISTON 8 SPARK PLUG 8 EXHAUST VALVE 8
DASHBOARD	2,000.00	
DIFFERENTIAL	500.00	
DRIVE SHAFT	500.00	
EXHAUST VALVE 1	47.50	
EXHAUST VALVE 2	47.50	
• • • •	• • • •	
EXHAUST VALVE 8	47.50	
FRAME	2,000.00	
INTAKE VALVE 1	47.50	
INTAKE VALVE 2	47.50	
• • • •	• • • •	
INTAKE VALVE 8	47.50	
PISTON 1	150.00	
PISTON 2	150.00	
• • • •	• • • •	
PISTON 8	150.00	
POWER TRAIN	5,000.00	ENGINE TRANSMISSION DRIVE SHAFT
SEATS	1,000.00	
SHELL	4,500.00	
SPARK PLUG 1	5.00	
SPARK PLUG 2	5.00	
• • • •	• • • •	
SPARK PLUG 8	5.00	
TRANSMISSION	1,000.00	
WATER JACKET	500.00	
WINDSHIELD	500.00	

FIGURE 6

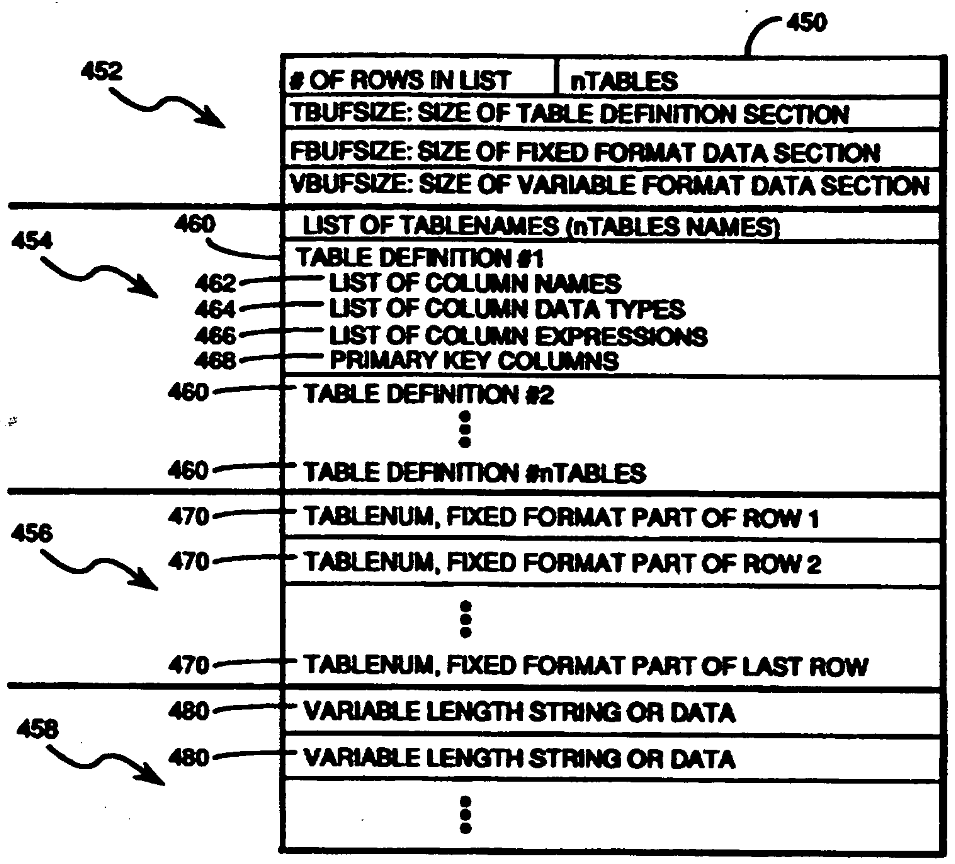

450
452
# OF ROWS IN LIST
nTABLES
TBUFSIZE: SIZE OF TABLE DEFINITION SECTION
FBUFSIZE: SIZE OF FIXED FORMAT DATA SECTION
VBUFSIZE: SIZE OF VARIABLE FORMAT DATA SECTION
454
460
LIST OF TABLENAMES (nTABLES NAMES)
TABLE DEFINITION #1
462 LIST OF COLUMN NAMES
464 LIST OF COLUMN DATA TYPES
466 LIST OF COLUMN EXPRESSIONS
468 PRIMARY KEY COLUMNS
460 TABLE DEFINITION #2
460 TABLE DEFINITION #nTABLES
456
470 TABLENUM, FIXED FORMAT PART OF ROW 1
470 TABLENUM, FIXED FORMAT PART OF ROW 2
470 TABLENUM, FIXED FORMAT PART OF LAST ROW
458
480 VARIABLE LENGTH STRING OR DATA
480 VARIABLE LENGTH STRING OR DATA

FIGURE 7

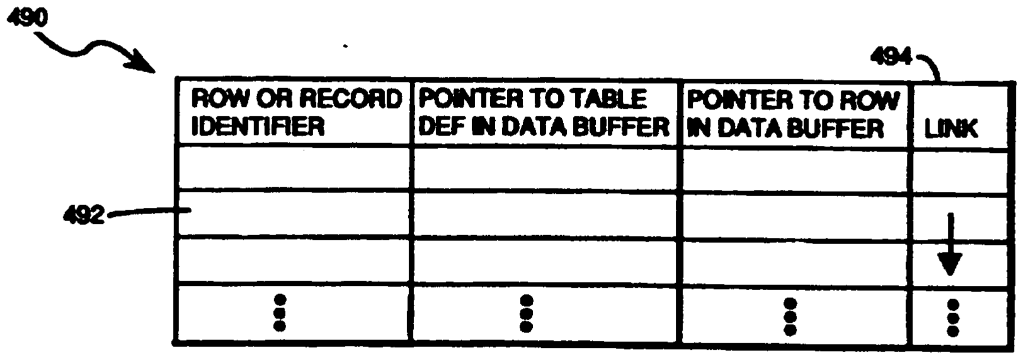

490
494
ROW OR RECORD IDENTIFIER
POINTER TO TABLE DEF IN DATA BUFFER
POINTER TO ROW IN DATA BUFFER
LINK
492

FIGURE 8

RELATIONAL DATABASE MANAGEMENT SYSTEM AND METHOD FOR STORING, RETRIEVING AND MODIFYING DIRECTED GRAPH DATA STRUCTURES

The present invention relates generally to computer database systems and particularly to relational database storage methods and systems for storing and retrieving directed graph data structures.

BACKGROUND OF THE INVENTION

A database management system (DBMS) is computer software that stores data and provides software routines for manipulating the stored data. A DBMS may be used directly (i.e., by human users), as a component of another software package, or to provide service to another software package.

A database is a collection of data which is stored and managed as a unit by a DBMS. A "relational database" is a database which contains tables that are used to store sets of data and to specify relationships between the different sets of data stored in the database. Relational databases and database management systems are widely used in the prior art. Therefore this document will describe prior art database systems only to the extent necessary to point out the differences between such prior art systems and the present invention.

Typically, databases are used to store sets of related data. For example, a database may be used to store all the seat reservations made by the customers of an airline, plus information about the airplane (e.g., seating chart information), information about the customers (e.g., address, credit card used to purchase tickets, and travel agent), and so on. This is an example of a database which is well suited for a prior art relational database management system.

The reason that the airline seat reservation database is easy to use with prior art database technology is that the data is easily organized as a set of flat records, in the form of a few tables: one for seat reservations, one for customer information, and so on.

An example of a set of data that is "difficult" to efficiently store and manipulate in a prior art relational

database is shown in FIG. 1. This set of data 100, which denotes a set of automobile parts and also denotes which parts are components of other parts, is herein called a "directed graph". The data structures conventionally used to store such sets of data in computers are called "directed graph data structures". The reason that a directed graph is "difficult" to handle with a conventional database system is that while this data can be stored in and retrieved from a conventional database table, it is awkward to do so.

FIG. 2 contains a typical prior art table 110 (herein called the ContainsParts table) that would be used by a prior art database management system to store the directed graph shown in FIG. 1. FIG. 2 also shows a second table 120 (herein called the Parts table) which contains cost data for automobile parts. By using the two tables 110 and 120 together, one can determine the relative costs of manufacturing various portions of an automobile.

While table 110 in FIG. 2 contains all the data needed to reconstruct the directed graph of FIG. 1, it is very awkward for a prior art database management system to utilize data which is organized in this fashion. For example, consider the steps which would need to be performed by the prior art DBMS to generate a directed graph representing the set of all components of the engine. To do this, we would first have to examine all the records with a partName of ENGINE to generate a first list of engine parts. Then we would have to examine all the records for the parts identified in this first search (i.e., with partName equal to CAM SHAFT or WATER JACKET OR CYLINDER 1, etc.). In a real life example, we would then have to examine all the records for the parts identified in the second search, and so on.

In terms of search commands using SQL, the industry standard language for querying databases, a separate query would be required for retrieving each set of subparts. As will be explained in more detail below, to regenerate the portion of the directed graph corresponding to ENGINE, one would have to perform literally dozens of queries. TABLE 1 lists the fifty-four SQL queries which would be required to regenerate the entire directed graph for AUTOMOBILE:

TABLE 1

PRIOR ART QUERIES FOR RETRIEVING DIRECTED GRAPH
1) SELECT * FROM ContainsParts WHERE PARTNAME = "AUTOMOBILE"
2) SELECT * FROM ContainsParts WHERE PARTNAME = "BODY"
3) SELECT * FROM ContainsParts WHERE PARTNAME = "FRAME"
4) SELECT * FROM ContainsParts WHERE PARTNAME = "POWER TRAIN"
5) SELECT * FROM ContainsParts WHERE PARTNAME = "DASH BOARD"
6) SELECT * FROM ContainsParts WHERE PARTNAME = "SEATS"
7) SELECT * FROM ContainsParts WHERE PARTNAME = "SHELL"
8) SELECT * FROM ContainsParts WHERE PARTNAME = "WINDSHIELD"
9) SELECT * FROM ContainsParts WHERE PARTNAME = "DIFFERENTIAL"
10) SELECT * FROM ContainsParts WHERE PARTNAME = "DRIVE SHAFT"
11) SELECT * FROM ContainsParts WHERE PARTNAME = "ENGINE"
12) SELECT * FROM ContainsParts WHERE PARTNAME = "TRANSMISSION"
13) SELECT * FROM ContainsParts WHERE PARTNAME = "CAM SHAFT"
14) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 1"
15) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 2"
16) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 3"
17) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 4"
18) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 5"
19) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 6"
20) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 7"
21) SELECT * FROM ContainsParts WHERE PARTNAME = "CYLINDER 8"
22) SELECT * FROM ContainsParts WHERE PARTNAME = "WATER JACKET"
23) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 1"
24) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 2"
25) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 3"
26) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 4"

TABLE 1-continued

PRIOR ART QUERIES FOR RETRIEVING DIRECTED GRAPH
27) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 5"
28) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 6"
29) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 7"
30) SELECT * FROM ContainsParts WHERE PARTNAME = "PISTON 8"
31) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 1"
32) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 2"
33) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 3"
34) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 4"
35) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 5"
36) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 6"
37) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 7"
38) SELECT * FROM ContainsParts WHERE PARTNAME = "SPARK PLUG 8"
39) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 1"
40) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 2"
41) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 3"
42) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 4"
43) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 5"
44) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 6"
45) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 7"
46) SELECT * FROM ContainsParts WHERE PARTNAME = "EXHAUST VALVE 1"
47) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 1"
48) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 2"
49) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 3"
50) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 4"
51) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 5"
52) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 6"
53) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 7"
54) SELECT * FROM ContainsParts WHERE PARTNAME = "INTAKE VALVE 8"

By way of comparison, the present invention allows a person or program to retrieve an entire subtree (or even a pruned subtree) of a directed graph using a single query. The single query needed to retrieve the entire directed graph in the preferred embodiment of the present invention is:

```
SELECT * FROM ContainsParts
        EXPAND ContainsParts(*)
        WHERE PARTNAME = "AUTOMOBILE"
```

The single query which would retrieve all portions of the directed graph corresponding to ENGINE is:

```
SELECT * FROM ContainsParts
        EXPAND ContainsParts(*)
        WHERE PARTNAME = "ENGINE"
```

Trees and other directed graph data structures are commonly used in scientific and engineering applications to represent and store data. Because of the limitations in the prior art, these types of scientific and engineering data are typically not stored using database management systems. As a result, all of the well developed tools associated with database management systems are generally not available to the users of scientific and engineering data. Instead, such data is typically stored and manipulated using a wide variety of special software programs. These programs vary widely in their manner of operation, how they represent data internally, and so on. Unlike relational database management systems, the programs each have a different theory of operation and each tends to be used by only a small market niche.

The primary goal of the present invention is to enable scientific and engineering data, which is normally stored in the form of tree data structures or directed graph data structures in operating system files (i.e., files directly accessed by application programs), to be easily stored and manipulated in a relational database management system. From another perspective, the primary goal of the present invention is to modify conventional relational database management systems so as to efficiently and intelligently handle data which is logically organized as a directed graph.

An important property of the present invention that is not provided by prior art relational database management systems is "transitive closure". Transitive closure means the ability to follow the links in a directed graph data structure and to process an entire or specified portion of a tree or directed graph as single entity. Database management systems which include the features of the present invention perform transitive closures, whereas prior art relational database management systems do not.

SUMMARY OF THE INVENTION

In summary, the present invention is an improved database management system (DBMS) which can store, retrieve and manipulate directed graph data structures in a relational database. Each directed graph data structure comprises one or more records of data which are interconnected by pointers. Data is stored in the database in the form of two dimensional tables, also known as flat files or base tables.

The improved DBMS defines a schema for each base table in the database. The schema defines the name and data type of each column in a database table. For tables used to store directed graph data structures, at least one column of the table will be defined as having a reference data type, which means that non-empty entries in that column contain "references" to other rows in the same or other tables in the database. A "reference" is a datum (stored in a reference column of a table) which matches the primary key of a particular row in a specified table in the database.

Directed graph data structures are stored in specified database tables by storing each record of the directed graph in a distinct row of a specified table. Interconnections between the directed graph's records are represented or denoted by references stored in reference columns, i.e., columns denoted in the table's schema as being a reference data type column. Portions of a directed graph are retrieved from base tables, in response to a query, by retrieving a portion of a first specified base table in accordance with the specified query and then expanding the retrieved data by also retrieving additional portions of base tables which are referenced by the portions of the first specified table already retrieved in accordance with the query.

In the preferred embodiment, the portions of a table retrieved in response to a query are stored in a buffer and then transmitted or communicated to an application process (i.e., to whomever sent the query to the DBMS). If the retrieved rows from the database include non-empty reference values, those reference values are automatically converted by the DBMS into pointers which point to other retrieved rows that are stored in the buffer. The application process includes an interface program for converting the retrieved portions of the specified table into a directed graph data structure.

Updates or modifications of directed graph data structures are handled in much the same way as storing a directed graph in the first place. The modified portions of the directed graph are converted into a set of rows in accordance with the specified schema for the target table or tables (i.e., the table(s) in which the directed graph is to be stored). Each resulting row of data is then used to modify or update corresponding portions of the target table(s).

BRIEF DESCRIPTION OF THE DRAWINGS

Additional objects and features of the invention will be more readily apparent from the following detailed description and appended claims when taken in conjunction with the drawings, in which:

FIG. 1 depicts an example of a directed graph.

FIG. 2 depicts two tables for storing data corresponding to the directed graph shown in FIG. 1.

FIG. 3 is a block diagram of a database management system in accordance with the present invention.

FIG. 4 illustrates a directed graph data structure.

FIG. 5 depicts the schema for a database table used to stored directed graph data structures.

FIG. 6 depicts an example of a database table used to stored directed graph data structures.

FIG. 7 depicts the data structure for a list of rows retrieved from a database using an expanded query.

FIG. 8 depicts a hash table used during data retrieval and storage.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring to FIG. 3, there is shown a multiuser database management system (DBMS) 200, drawn so as to emphasize the portions of the DBMS which are particularly relevant to the present invention. Before discussing this Figure in detail, the following glossary of terms is provided.

GLOSSARY

APPLICATION—A computer program that solves some particular problem. Applications may use a DBMS to store and manipulate information relevant to such application.

APPLICATION PROGRAMMER—The person or persons who write the source code for an application, such as a computer aided design program. Application programmers of ordinary skill in the art possess the skills necessary to implement applications within a specified application domain, and are also skilled in the use of prior art RDBMS products such as IBM's DB2, Oracle's Oracle, and Ingres (a trademark of Ingres Inc.). Application programmers of ordinary skill in the art are also skilled users of programming languages, such as C, the MAINSAIL programming language, and ADA, and can construct programming language interfaces which enable an application to send queries to a prior art relational DBMS system and to receive data and other responses generated by the prior art DBMS.

BASE TABLE—A table that corresponds to data physically stored in a permanent storage medium, as distinct from a derived table that is defined by a query used to retrieve data from a base table.

C—A programming language, often used to write engineering application programs. See for example, B. Kernighan and D. Ritchie, "The C Programming Language", 2nd Edition, Prentice Hall, New Jersey, 1988.

CYCLE—In the context of structured data such as directed graph data structures, a cycle is a closed loop consisting of a series of references from one component back to itself. For example, if the interstate highway system were represented in a database, where each highway was a row that contains references to each other row (highway) for which there is an exit, there would be many cycles, because it is possible to start on one highway, exit to another, and eventually exit back onto the original highway.

DATA ITEM—An instance of a particular data type. For example, the number 1 is an instance of the data type INTEGER.

DATA TYPE—A name for a set of possible values that may be represented in a computer's memory, along with a set of operations on those values. For example, the data type INTEGER allows for the representation of positive counting numbers (1,2,3,...), negative numbers (−1,−2,−3,...), and zero (0), and provides for arithmetic operations such as addition and multiplication. Each DBMS product supports a predefined set of data types.

DATABASE—A collection of related information, or data. Many databases are abstract representation of information pertinent to an enterprise, such as the design of integrated circuits.

DATABASE MANAGEMENT SYSTEM (DBMS)—A component of a computer system that provides support for the storage and management of databases.

DERIVED TABLE—A table that is defined by the results of a query that retrieves data.

DIRECTED GRAPH DATA STRUCTURE—A data structure that models arbitrary relationships among data objects. Directed graph data structures include, for example, trees, linked lists, and cyclical graph structures (structures with cycles). Directed graph data structures are defined and discussed in Aho, Data Structures and Algorithms, Addison-Wesley, 1983 (pp. 198-199).

ENGINEERING APPLICATIONS—Applications, such as computer-aided design (CAD), computer-aided manufacturing (CAM), and computer-aided software engineering (CASE), that must retrieve, manipulate, and store structured data that model objects of much greater complexity than typically found in business applications.

FOREIGN KEY—A foreign key in a table T1 is a list of one or more columns in table T1 that correspond to the primary key columns of a base table T2. (Table T1 and T2 may or may not be distinct.) Thus, the foreign key column values in a particular row of T1 logically refer to the row of T2 whose primary key column values matches such foreign key column values.

LIST—An ordered collection of zero or more data items, possibly containing some duplicate values.

NULL—A missing or unspecified value.

PRIMARY KEY—Associated with each base table in a relational database is an ordered list of columns whose values uniquely identify a particular row of the table. This list of columns comprises the primary key of the table. More particularly, the primary key of an individual row of the table consists of the values of the primary key columns for that row. Specification of the primary key values for a given row is the only way to identify that particular row.

PROGRAM—See application.

QUERY—An instruction (command) to a DBMS that tells the DBMS to perform a desired action. Typical actions include the retrieval of data from the database and making changes to the data stored in the database.

QUERY LANGUAGE—A computer language for expressing data retrieval questions and modification operations on a database. The query language defines the set of commands that are accepted by the DBMS, including the syntax and the actions or meaning associated with the syntax. A query is thus a particular instruction or sentence in the query language.

RECORD—A record is a memory resident aggregate data type supported by many programming environments. Each such programming environment provides a way to declare a description for each type of record used by an application (including the names and types of the constituent data types making up the record), a way to allocate new instances of each record type, and, often, a way to dispose of (recycle) records when no longer needed by the application.

REFERENCE—REFERENCE is a database column data type which is an extension of the concept of foreign key. Column that are declared in the schema to be of type REFERENCE are used to refer to individual rows by matching foreign key values to primary key values. The REFERENCE data type is used in the database representation of directed graph data structures. In a table having a reference column, each nonempty reference in the table matches the primary key value of a row in a specified table in the database. References differ from foreign keys in several ways that are explain in detail below. Most importantly, a reference is essentially a set of bits which match the primary key of a row in a base table. Each reference value is stored in a single reference column, even if the corresponding primary key occupies two or more columns of the referenced table.

RELATION—A data table of values, such data table being an open-ended and unordered collection of rows, each row consisting of an ordered and fixed list of data items.

RELATIONAL DBMS (RDBMS)—A DBMS that allows an application to define and operate on a database using the abstractions of the standard relational model, which frees the application from physical database storage considerations. In particular, data are represented using relations (tables).

RETRIEVAL QUERY—Any query that specifies that data be retrieved (passed from the DBMS to an application).

ROW—The list of related column values in a table corresponding to a particular primary key value.

SCHEMA—A description of the information that is represented by a database. More particularly for relational databases, a description of the tables that make up the database, including the table names, the column names and column types of each table, and any other information that is needed to enable an application, a DBMS, or a user to interpret the contents of the database.

SET—An unordered collection of zero or more unique data items, none of which can be null.

STRUCTURED DATA—Data that model the complex structure of real-world objects and events, including (in general) multiple links from one component of the structure to another and cycles. Structured data may conveniently be represented using a directed graph data structure.

STRUCTURED QUERY LANGUAGE (SQL)—The industry-standard query language for relational DBMS, as defined by the American National Standards Institute's standard ANSI X3.135-1986. An extension of this standard to include more powerful constructs, called SQL2, is currently being carried out in the working joint standards committee ANSI X3H2 and the International Organization for Standardization ISO DBL SYD-2. NOTE: The use of the word "structured" in the context of SQL is completely unrelated to the use of the same word in "structured data".

TABLE—Each table is described by the database schema and consists of a set of "rows", each of which contains values for each column of the table.

TOP-LEVEL RECORD—The starting point of a directed graph data structure as represented in an application's memory.

TOP-LEVEL ROW—The starting point of a directed graph data structure as represented in an extended relational database.

DATABASE MANAGEMENT SYSTEM

The database management system (DBMS) 200 shown in FIG. 3 is coupled to a number of application processes 202-204, hereinafter called applications. The DBMS 200 stores, retrieves and updates information in a database 206 on behalf of the applications. One of the primary benefits of using a DBMS 200 is that it relieves applications programmers from having to deal with data storage and retrieval, and instead allows application programmers to concentrate on solving the problems for which a specified application program is being developed. Another benefit of using a DBMS 200 is that it provides a mechanism which allows multiple applications to share the information in a shared database 206.

Physically, the data in the database is typically stored partially in high-speed random access memory (RAM) and partially on disk drives. A storage management subsystem 208 manages physical storage of the database 206, and typically includes performance enhancement features such as software for disk caching and for indexing on column values. A "b-tree" or "b+tree" physical storage technique is often used by commercial DBMS. Since storage management is not related to the present invention, and is well understood by those skilled in the art, it is not discussed any further herein.

In the preferred embodiment, each application process 202, 204 will reside on a separate computer, each of which is coupled to a host computer or network server which runs the database management system process 210. The DBMS process 210 includes a separate application task 212-214 for each application that is currently using the database. To coordinate communications with multiple applications and to allow a single computer to run multiple application tasks, the DBMS process 210 includes a multitasking transaction manager 220. In other words, the computer on which the DBMS is running includes multitasking operating system software for handling multiple tasks or execution threads.

As will be understood by those skilled in the art, there are many possible system configurations, all of which are equivalent for the purposes of the present invention. For example, in other embodiments it would possible for both the DBMS and the application programs to all be running on a single computer, or for the DBMS and some of the application programs to be running on one host computer (e.g., a mainframe or high performance minicomputer) with other application programs running in separate computers that are coupled to the host computer by a communications network.

The commands sent by application processes to the DBMS 200 are typically called queries. Furthermore, many commercial database systems use an industry standard query language, called SQL (Structured Query Language), which defines the syntax and interpretation of the queries which can be used by applications to store, retrieve and manipulate data in a database system. For the purposes of this description, it is assumed that queries sent by applications to the DBMS 200 in the preferred embodiment of the present invention conform to SQL, with a few exceptions that are described in detail below.

Each application task 212-214 in the DBMS process 210 uses a shared set of software for handling queries from applications. The three primary software modules or sets of software which are modified by the present invention are the software 222 for defining database tables and their associated schemas, the software 224 for interpreting queries sent by application programs, and the software 226 for handling data storage, retrieval and modification.

The data stored in the database 206 is organized as a set of base tables. Each base table 227 consists of an array of data organized in columns and rows. For each base table 227 there is a corresponding schema 228, which defines the data type associated which each column in the table 227. For instance, the first column of the table 110 (shown in FIG. 2) has a data type of "character string" and the second column also has a data type of "character string". The second column of table 120 shown in FIG. 2 has a data type of "decimal". Base tables and schemas will be described in more detail below.

As is standard in prior art DBMS's, each base table 227 also has one or more indices 229 which are specialized data structures for quickly locating any specified row in a base table. As will be discussed below, every base table 227 has a primary key and an index for that primary key.

APPLICATION INTERFACE.

In the preferred embodiment, each application process 202-204 includes an application program 230 and an application interface 232, as shown in FIG. 3. The

purpose of the application interface 232 is firstly to communicate queries from the application program to the DBMS process 210 (see software module 234) and secondly to communicate or transmit data in both directions between the application program 230 and the DBMS process 210. In the context of the present invention, the software 236 for transmitting data between the application program 230 and the DBMS process 210 is called a data translator or converter because it converts directed graph data structures 240 from the application program 230 into a form suitable for transmittal to the DBMS process 210, and also converts data retrieved from the database 206 into directed graph data structures for use by the application program 230.

More specifically, data retrieved from the database 206 in response to a query from application 202 is transmitted to the application process 202 in the form of a list 244 of one or more rows of retrieved data. The list 244 contains portions of one or more of the base tables stored in the database 206. In other words, each row in the list 244 comprises a selected row, or a portion of a selected row from a base table. Furthermore, each row in the list 244 may come from a distinct base table.

The list of rows 244 retrieved from the database 206 is temporarily stored in a buffer 241 in the application task 212 of the DBMS process, and then the contents of that buffer are transmitted to buffer 242 in the application interface 232. The data translator software 236 converts the retrieved data stored in the buffer 242 into a directed graph data structure for use by the application program 230. In other words, each row of the list 244 in the buffer is converted into a record with a data structure compatible with the application program 230.

When directed graph data structures 240 are sent by the application program 230 to the DBMS process 210 for storage in the database 206, the data translator 236 in the application interface 232 converts that data into a list of rows 246, and stores that list in buffer 242. Each row in the list 246 has an associated row number. The pointers in each record (i.e., pointer fields which point to other records) that is stored as a row in list 246 are converted into row numbers. Furthermore, each row or record in list 246 has a set of columns that are a subset of the columns of a base table that is stored in the database 206. After the records of the directed graph have been converted into a list of records stored in buffer 242, the contents of the buffer 242 are transmitted to the DBMS process 210 for storage in one or more target base tables. In the most common case, all the columns in each record of the list 246 match the data types of columns in a target base table, the records in the list 246 can be directly copied into the target base table.

The data structure for storing the retrieved list of rows 244 and the list of records to be stored 246 is described below with reference to FIG. 7.

DIRECTED GRAPH DATA STRUCTURE

The present invention is particularly suited for use in conjunction with engineering application programs, such as computer-aided design (CAD) programs, computer-aided manufacturing (CAM), and computer-aided software engineering (CASE), and other programs which must store, retrieve, and manipulate structured data that model objects of much greater complexity than typically found in business applications.

Referring to FIG. 4, there is shown a directed graph data structure 330, similar to FIG. 1, but explicitly showing the pointers between records in the data structure in one particular programming environment. Each record 332-346 of the directed graph data structure 330 in this example includes a name field 350, followed a decimal number field 351 for storing cost data, followed by a set 352 of pointers herein labelled CONTAINSPARTS[1], CONTAINSPARTS[2], and so on. For example, the top-level record 332 of the data structure 330 has a name field with a value of AUTOMOBILE, and is coupled to records 334, 336 and 338 by a set of three pointers. Record 334 is coupled, in turn, by four pointers to records 340, 342, 344 and 346. Record 340, labelled ENGINE, is coupled by pointers to ten other records, not shown in this Figure.

SCHEMA AND TABLE FOR STORING DIRECTED GRAPHS.

The primary change to the software 222 for defining tables and their associated schemas is simply to add one additional data type to the set of allowed data types which can be specified for the columns of a table. In particular, the present invention adds a new data type herein called the "Reference" data type, for columns of a table which contain references to other rows in either the same base table or another base table.

Referring to FIGS. 5 and 6, there is shown a base table 400, herein called the "parts" table, which is suitable for storing the directed graph data structure shown in FIG. 4, and the schema 410 for that base table. In this particular example, the base table 400 shown in FIG. 6 has three columns: a "partname" column 402, a costs data column 404, and a "containsparts" column 406 containing a set (see definition of a "set" in the Glossary, above) of references. The base table 400 has a multiplicity of rows 420, each of which contains a related collection of data (in this example, the data in each row concerns one "part" of an automobile).

The schema 410 for a table denotes the name of the table, and then contains a distinct entry 412 for each column of the table 400, each entry defining the name and data type of one column in the corresponding base table 400. The first item 414 in the schema 410 in FIG. 5 is the table's name, which is "Parts". The next item 416 in the schema defines the first column 402 of the table 410, including the name of the column, "PartName", and its data type, which is "character string". Next, the schema contains a definition 418 of the second column of the table 404, which stores cost data. As shown by the schema item for this column, the column has a name of "Cost" and has a data type of "decimal(9,2)", which indicates the cost data contains nine digits, two of which are to the right of a decimal point. The last entry 420 in this schema 410 indicates that column 406 is a SET (which is an array containing an unordered collection of non-empty unique values) having the name "CONTAINSPARTS", with a data type of "Reference (Parts)", which means that each value stored in the CONTAINSPARTS SET is a reference to a row in the Parts table 404. Reference data type columns will be described in more detail below.

To define a new base table in a database, the DBMS provides a CREATE command which defines the name of the table, the name and data type of each column in the table, and also the primary key associated with the table. For example, the CREATE command for creating the table in FIG. 6 would be:

```
CREATE TABLE Parts (
```

-continued

```
STRING partName;
DECIMAL(9,2) cost;
REFERENCE (Parts) SET containsParts;
PRIMARY KEY partName )
```

Execution of this command would generate the schema 410 shown in FIG. 5 and an empty table 400 having the format of the table shown in FIG. 6. While the primary key in this example is based on a single column of the table, in many applications, the primary key will be an ordered list of columns whose values uniquely identify a particular row of the table.

As is standard in prior art DBMS's, tables and their schema can be modified after their initial definition. Thus extra columns can be added to a pre-existing base table and columns can be deleted. For example, it is possible to add columns with a Reference data type to a pre-existing base table which does not contain any Reference columns.

The preferred embodiment of the present invention also adds three additional data types not found in prior art relational database management systems: ARRAY, LIST and SET. In particular, any column of a base table can be defined to have a data type of ARRAY, LIST or SET, as well as the other standard data types, such as INTEGER, STRING, DECIMAL(x,y) and so on. An ARRAY is simply a conventional array, such as the arrays used in FORTRAN and other computer languages. What is unusual about the ARRAY data type in the context of the present invention is that a column of a base table can be defined as an array, and thus each row in the base table will store an array of data in that particular column. A LIST is an ordered collection of data, which may contain data that is null or non-unique. A SET is a unordered collection of non-empty unique data values. Prior art relational database management systems could only store one data value in each column position of a row. These three new data types make it much easier to store the type of data encountered in engineering and scientific type applications.

REFERENCE COLUMNS AND POINTERS

The data stored in a column with a data type of Reference are indirect pointers or "references" to rows in a specified base table. As shown in FIG. 5 above, the specification for a reference column in a table's schema specifies not only the data type "Reference" but also the name of the base table for which the column contains references. Thus, all the references in one reference column of a table may reference rows in "Table A" while the references in another reference column will reference rows in "Table B". Table A or B may, or may not, be the same table as the one containing the references.

More generally, references are similar to "foreign keys". A foreign key, as used in prior art DBMS's, is an ordered set of column values stored in a first table which matches the primary key value of a row in another table. Each non-empty value stored in a single reference data column, on the other hand, matches the primary key value of a row either in the same table as the table containing the reference data column, or in another base table, regardless of the number of columns required to define that primary key. In other words a reference value stored in a single column will match the binary bit pattern of a multiple column primary key. The values stored in Reference columns are sometimes herein called "pointers" because primary key values are, logically, pointers to rows in a table. In terms of DBMS programming techniques, each primary key is converted during the course of transaction processing into a pointer to a particular address in a computer's memory through the use of an index. Thus each non-empty reference value points, directly or indirectly, to a particular row in a specified table.

STORING A DIRECTED GRAPH IN A BASE TABLE

The first step in storing a directed graph data structure in a database is for the application program 230 to transmit the directed graph data structure to the application interface 232. Note that when a directed graph is being sent by an application for storage in a database, there must already be one or more base tables 227 in the database 206 which have suitable sets of columns for storing this directed graph. Note that a directed graph may contain a number of different types of records, each of which stores different types of data. Typically, each type of record in the directed graph will be stored in a different base table, although other arrangements are possible.

If the base tables needed to store the directed graph do not yet exist, the application program 230 must first send instructions (i.e., a CREATE TABLE command) to the DBMS 210 so as to define the needed base tables. These base tables are called the "target" base tables in the database.

Intermediate Data Format for Buffered Data

Referring to FIG. 7, data is transported in both directions between the application program and the database management system using a "portable data representation", which is essentially a self-documenting list of rows. FIG. 7 shows the data structure 450 of the transported data as stored in the buffer 242 of FIG. 3. Thus the terms "buffer", "portable data structure", and "intermediate data structure" are herein used interchangeably.

As shown in FIG. 7, the data stored in the buffer 242 has a header 452, a table definition section 454, a fixed format row storage section 456, and a variable length format data section 458. The header 452 defines the size of the buffer and the size of each section of the buffer, as well as the number of rows of data stored in the data structure.

The table definition section 454 defines the format of each distinct type of row stored in sections 456 and 457. The first item in this section is a list 459 of all the tables names for which table definitions follow. Each table definition 460 includes a definition for each column, including a list of column names 462, a list of column data types 464, and for those columns which contain derived data, expressions 466 denoting how values in the columns were derived. The latter field 466 is used only when retrieving data from the database. There is also a primary key definition 468 denoting the ordered set of columns of the table which form its primary key.

Usually, when data from a directed graph is being inserted into a database, the columns in each row which is stored in the intermediate format are the same as, or a subset of the columns of the target base table in which the directed graph is being stored. However, an update query can specify an expression defining the value to be stored in particular column as a specified function of one or more fields in the directed graph record being updated.

The fixed format section 456 contains one row 470 for every row of data that is being transported. Each row 470 begins with a "table number", which references one of the table definitions in section 454. The remainder of the row 470 contains all the columns of the row which are fixed length data items, i.e., excluding variable length strings, lists, arrays and so on. Each column in the row 470 which contains variable length data is replaced by a pointer to an item in section 458 of the buffer, at which position is stored the actual data value for that column. Furthermore, for each reference column, the reference value is replaced by a row number indicating which row in the fixed format data section 456 is being referenced.

Note that by replacing variable length data with fixed format pointers, the rows in section 456 are all fixed in length.

The variable format section 458 contains variable length strings and other variable length data. Each such item 480 stored in this section 458 is pointed to by a column entry in one of the rows 470.

Furthermore, when storing data into the buffer format shown in FIG. 7, either while retrieving data from the database or storing a directed graph for transportation to the database, the following technique is used to avoid storing duplicate copies of rows or records. Referring to FIG. 8, for each row stored in the buffer, an entry 492 is made in a hash table 490. That entry 492 contains a unique row or record identifier (e.g., the primary key for rows retrieved from the database, and the address of records obtained from a directed graph), a pointer to the corresponding table definition in the table definition section 454 of the buffer, and a pointer to the row as stored in the fixed format data section 456 of the buffer. It also contains a link field 494 for sequentially accessing all entries in the hash table in the same order that the entries were added to the hash table. Before storing each row or record in the buffer 241 or 242 the application interface or DBMS (depending on which direction data is being transported) checks the hash table 490 to see if there is already an entry 492 in the hash table for that row or record. If so, the row or record has already been stored in buffer 241 or 242 and so the row or record is not stored a second time in the buffer.

Updating a Specified Portion of a Base Table

In the preferred embodiment, using an extended version of the SQL query language, the modification queries include an INSERT statement, an UPDATE statement, and a DELETE statement.

Consider storing the rows in parts Table 400 (shown in FIG. 6) related to the engine and parts within the engine. Using prior art SQL commands requires 43 queries:

```
INSERT  INTO  Parts  VALUES   ("ENGINE", 3000.00, "CAM SHAFT", . . . )
INSERT  INTO  Parts  VALUES   ("CAM SHAFT", 500.00, "", . . . )
```

-continued

```
INSERT  INTO  Parts  VALUES  ("WATER JACKET", 500.00, "", . . . )
INSERT  INTO  Parts  VALUES  ("CYLINDER 1", 250.00, "PISTON 1", . . . )
INSERT  INTO  Parts  VALUES  ("EXHAUST VALVE 1", 47.50, "", . . . )
INSERT  INTO  Parts  VALUES  ("INTAKE VALVE 1", 47.50, "", . . . )
INSERT  INTO  Parts  VALUES  ("PISTON 1", 150.00, "", . . . )
INSERT  INTO  Parts  VALUES  ("SPARK PLUG 1", 5.00, "", . . . )
.       .     .      .       . . .
.       .     .      .       . . .
.       .     .      .       . . .
INSERT  INTO  Parts  VALUES  ("CYLINDER 8", 250.00, "PISTON 8", . . . )
INSERT  INTO  Parts  VALUES  ("EXHAUST VALVE 8", 47.50, "", . . . )
INSERT  INTO  Parts  VALUES  ("INTAKE VALVE 8", 47.50, "", . . . )
INSERT  INTO  Parts  VALUES  ("PISTON 8", 150.00, "", . . . )
INSERT  INTO  Parts  VALUES  ("SPARK PLUG 8", 5.00, "", . . . )
```

Using the schema definition for Table 400 and the INSERT statement provided by the preferred embodiment of the present invention, the forty-three rows of Table 400 that describe the engine and its subparts are inserted using the single query:

INSERT INTO PARTS USING ARG

where ARG refers to a directed graph data structure argument, supplied by the application program, that represents the engine and its subparts. In other words, the application program would first construct a directed graph data structure called ARG. Then it would execute the above INSERT query to insert all the engine data into a specified table.

Thus, to insert all the data about the engine and its subparts using this invention requires only the single query, in contrast to the forty-three queries of the prior art.

Syntax for Insert Statement

Examples of query language extensions are specified below using a modified form of "Backus-Naur Form" (BNF), a notation commonly used to describe computer programming languages. Statements are described by giving their syntax in terms of entities and keywords. Entities are represented as a word enclosed in angle brackets ("<" and ">"). Keywords are shown as capitalized words not enclosed in brackets. Bracketed entities are defined using the symbol "::=". Square brackets ("[" and "]") enclose optional portions of the syntax. If a right square bracket is followed by an asterisk ("*"), the entities and keywords within the brackets may occur zero or more times. An asterisk that does not follow a right square bracket is taken literally.

The Query command used in the preferred embodiment for inserting or storing data into the database has the following format, denoted in modified "Backus-Naur Form":

```
<insertStatement>
        ::= INSERT [INTO] <tableName> [( <insertList> )] <insertSpec>
<insertList>
        ::= <columnName> [, <columnName>]*
<insertSpec>
        ::= VALUES <valuesSpec> [, <valuesSpec> ]*
        ::= SELECT [ALL | DISTINCT] <selectList>
            FROM <tableExpression> [ WHERE <searchCondition> ]
        ::= USING ARG [ <valuesSpec> ]
            [ WHERE <searchCondition> ]
<valuesSpec>
        ::= <valueExpression> [, <valueExpression> ]*
<tableExpression>
        ::= <tableReference> [,<tableReference>]*
```

Thus an INSERT statement is used to insert a directed graph, or specified portions of a specified directed graph, into a specified base table in the database. The USING ARG portion of the statement defines what portions of the ARG directed graph data structure are to be inserted into the database. In particular, for each record REC in the ARG directed graph data structure whose corresponding base table is <tableName> and for which <searchCondition> is true (i.e., when applied to REC), the application interface inserts into the columns specified by <insertList> of the base table specified by <tableName> values of the fields in REC given by <valuesSpec>. If the optional <valuesSpec> in the USING ARG clause is omitted, the fields of the inserted records are assumed to correspond to columns of <tableName> having the same names.

Syntax for Update Statement

The UPDATE statement updates data in the columns of pre-existing rows in a specified base table. The Query command used in the preferred embodiment for updating data previously stored in the database has the following format, denoted in modified Backus-Naur form:

```
<updateStatement>
        ::= UPDATE <tableName>
            [ USING ARG [AS <correlationName> ] ]
            <updateSpecification> [, <updateSpecification>]*
            [ WHERE <searchCondition> ]
```

This statement is interpreted as follows: For each record REC in the directed graph data structure whose corresponding base table is <TableName>, locate the row R of table <tableName> whose primary key matches the primary key of REC. If <searchCondition> is true (i.e., when applied to REC and/or R), update row R as specified by the <updateSpecification>s. An example of an <updateSpecification> is

"SET A=NEW.A". If the optional <correlationName> is not given, its default value is "NEW". The fields of REC may be referred to in <searchCondition> using

```
<correlationName>.<fieldName>
```

An example of an UPDATE statement is:

```
UPDATE T1
USING ARG
SET a = new.a, b = new.a * 100
```

SYNTAX FOR DELETE STATEMENT

The Query command used in the preferred embodiment for deleting data previously stored in the database has the following format, denoted in modified Backus-Naur form:

```
<deleteStatement>
     ::= DELETE FROM <tableName> <deleteSpec>
<deleteSpec>
     ::= WHERE <searchCondition>
     ::= ALL
     ::= USING ARG [AS <correlationName> ]
         [ WHERE <searchCondition> ]
```

The DELETE statement is interpreted as follows: For each record REC in the ARG directed graph data structure, locate the row R of table <tableName> whose primary key matches the primary key of REC. If <searchCondition> is true (i.e., when applied to row R and to record REC), delete row R from <tableName>. If the optional <correlationName> is not given, its default value is "NEW". Fields of REC may be referred to in <searchCondition> using

```
<correlationName>.<fieldName>
```

An example of a DELETE statement is:

```
DELETE FROM T1
USING ARG
```

CONVERTING A DIRECTED GRAPH INTO INTERMEDIATE FORMAT

The following description includes pseudocode representations of the software routines relevant to the present invention. The pseudocode used herein is, essentially, a computer language using universal computer language conventions. While the pseudocode employed here has been invented solely for the purposes of this description, it is designed to be easily understandable by any computer programmer skilled in the art. The computer programs in the preferred embodiment are written primarily in the MAINSAIL programming language, compilers for which are commercially available from Xidak, Inc. of Palo Alto, California.

Referring to FIGS. 3, 7 and 8, the procedure used by the data translator 236 in the interface 232 to convert a specified directed graph data structure into the intermediate data format shown in FIG. 7, is as follows.

PSEUDOCODE FOR STORING DIRECTED GRAPH IN BUFFER

```
INITIALIZATION:
    CLEAR HASH TABLE          - (see FIG. 8)
    CLEAR LIST RECS FOR STORING RECORDS
    CLEAR LIST OF TABLE DEFINITIONS TDEF
STORE IN TDEF THE TABLE DEFINITIONS FOR ALL DISTINCT TYPES
OF RECORDS THAT ARE TO BE INSERTED INTO THE DATABASE
STORE TOP LEVEL RECORD IN RECS
STORE CORRESPONDING ENTRY IN HASH TABLE
FOR EACH RECORD IN RECS (BEGINNING WITH TOP LEVEL RECORD)
    FOR EACH POINTER IN RECORD
        RR = RECORD POINTED TO BY POINTER
        IF RR IS NOT ALREADY IN HASH TABLE AND
          RR MEETS SPECIFIED <searchCondition>
            ADD RR TO HASH TABLE
            ADD SPECIFIED FIELDS OF RR TO RECS
        ENDIF
    ENDLOOP
    USE LINK IN HASH TABLE TO FIND NEXT RECORD, IF ANY
ENDLOOP
STORE TDEF AND RECS IN BUFFER FORMAT SHOWN IN FIG. 7
```

The initialization step in the above pseudocode routine sets up an empty hash table and two lists: one for records called RECS and one for table definitions called TDEF. Table definitions for all the distinct types of records that are to be inserted into the database are added to the TDEF list.

The list of records in the application interface buffer 242 is constructed by visiting each record in the directed graph data structure once, and recording its associated information in the buffer. More specifically, every record reachable from the specified starting record is processed.

To prevent storing a record more than once, which would be the case when the directed graph contains cycles or multiple pointers to one record, the application interface checks, upon visiting each potentially new record, whether that record is already in the RECS list (by looking for the record in the hash table 490), and if so, it does not process that record any further. If the record has not already been visited, the application interface checks the specified <searchCondition> to determine whether to add this record to the RECS list. If so, the fields of the record specified by <valuesSpec> are stored as a new row in the RECS list. The record is also added to the hash table. Then the buffer building process moves onto the next record found using the link field of the hash table.

When all the records of the directed graph have been visited, the resulting lists TDEF and RECS are stored in the application interface buffer 242 in the format shown in FIG. 7, which was described above.

Next, the list of rows 246 in buffer 242 is transmitted by the application interface 232 to the corresponding application task 212 in the DBMS process 210, along with the command (i.e., INSERT, UPDATE or DELETE) which is transmitted to the DBMS by software module 234 in the application interface.

The list of rows 246, now in buffer 241 of the application task 212 in the DBMS, are then either added to the specified target base table, or used to update pre-existing records in the target base table (if the application's query was an UPDATE or DELETE command). During this process, the Reference pointers stored in the intermediate buffer format as row numbers are replaced with the primary key values of the appropriate rows in the target base table.

When a modification query that contains a USING ARG clause is issued by an application, the application must provide additional declarative information to the application interface to enable it to traverse through the directed graph data structure, and to enable it to associate each type of record in the directed graph data structure with a particular table in the database. This additional information is provided along with both the query and the directed graph data structure (ARG) itself.

For programming languages, such as the C language, that do not represent data structures in a self-descriptive way (at runtime), the additional declarative information must include the following information for each distinct type of record in the directed graph data structure that is to be sent to the DBMS:

1) The name of each field in this record type that is to be passed to the DBMS.
2) The displacement of each such field from the start of the record.
3) The type (e.g., integer, string, decimal, ...) of each such field.
4) If the field is a pointer to a record, the name of the database table corresponding to the pointed-to record.
5) The name of the table in the database to which this record type corresponds.

In addition, the record description that corresponds to the top-level record in the directed graph data structure must be identified so that the application interface knows how to "begin" its traversal of the directed graph.

For programming languages that have self-descriptive data structures, such as the MAINSAIL programming language, the additional declarative information must include the following information for each distinct type of record in the directed graph data structure that is to be sent to the DBMS:

1) An instance of this type of record, referred to as a "model record".
2) The name of the table in the database to which this type of record corresponds.

The model record is used as a template against which records encountered in the traversal of the directed graph data structure can be matched. For each record REC encountered, the application interface compares the REC's type with each of the model record type's

until a match is found. It then knows the name of the table in the database that corresponds to REC.

Another feature of the preferred embodiment, which improves efficiency by reducing the number of distinct queries to be processed by the DBMS, is that the protocol between the application interface and the DBMS allows more than one query command to be transmitted as a part of a single query string. More particularly, a single transmitted query string can contain several query commands separated by semicolons (i.e., "query1; query2; ..."). Thus, a multiplicity of INSERT, DELETE and UPDATE statements may be associated with a single directed graph data structure.

Thus, for both self-descriptive and non-descriptive application programming languages, the declarative information passed to the application interface along with the ARG includes a description of each distinct type of record in the directed graph data structure, and the name of the corresponding database table.

RETRIEVING A SPECIFIED PORTION OF A DIRECTED GRAPH

In general, data is retrieved from any database by sending a query to the DBMS 200. The DBMS 200 interprets the query, generates a set of detailed instructions for executing the specified query, and either returns the requested data or an error code that explains why it is unable to comply with the query.

For instance, referring to the table 400 shown in FIG. 6, an example retrieval query using prior art SQL would be:

```
SELECT *
FROM parts
WHERE partName = 'ENGINE'
```

When applied to table 400 in FIG. 6, this query requests the DBMS to retrieve the row of the base table whose partName is "ENGINE". This prior art query retrieves only one row of the table: the row with a partName of "ENGINE". Of course, another prior art query could be used to retrieve all the rows of the table 400 which have a PartName beginning with the string "PISTON", which would result in the retrieval of eight rows: the rows for PISTON 1 through PISTON 8.

More importantly, the prior art query for retrieving the "ENGINE" row does not retrieve any information about the ten parts which are components of the engine, except for their names. Note that this discussion assumes that a prior art version of base table 400 would have the names of the engine's components stored in the "ENGINE" row of table 400 instead of references to the rows of table 400 for those components.

To solve this problem, the present invention modifies the industry standard SQL language to include two new keywords, EXPAND and DEPTH, for controlling the retrieval of directed graph data structures. In the preferred embodiment, whenever a SELECT statement includes the keyword EXPAND (and does not include the keyword DEPTH), the DBMS process will respond to the SELECT query by retrieving two sets of data:

(1) all rows, or specified portions of rows, which meet a specified set of conditions, which are typically denoted by a WHERE or HAVING clause in the SELECT statement; and

(2) all rows, or specified portions of rows, which are pointed to by Reference pointers in other rows of data retrieved in response to the query.

It is important to note that the second category of data which is retrieved is a recursive definition. In other words, the DBMS process continues to retrieve data until all rows which are pointed to by any previously retrieved row have been retrieved.

The purpose of the DEPTH <n> clause in the SELECT statement is to limit the amount of data retrieved by a SELECT statement which includes the EXPAND keyword. In particular, whenever a DEPTH <n> clause is used, every row of the specified table which is retrieved must by connected to one of the retrieved top-level row(s) by a chain of n−1 or fewer pointers.

In the preferred embodiment, the extended SQL syntax for query select statements provided by the present invention, denoted in modified "Backus-Naur Form" is:

```
<selectStatement>
        ::= SELECT [DISTINCT | ALL] <selectList> FROM <tableExpression>
            [ ORDER [BY] <sortColumn> [,<sortColumn>]* ]
<selectList>
        ::= <valueExpression> [AS <columnName>] [, <selectList>]*
        ::= <qualifier>.* [, <selectList>]*
        ::= *
<tableExpression>
        ::= <tableReference> [,<tableReference>]*
            [ EXPAND [DEPTH <n>] <expansionSpec> [, <expansionSpec>]* ]
            [ WHERE <searchCondition> ]
            [ GROUP [BY] <columnSpecification>
                  [, <columnSpecification>]* ]
            [ HAVING <searchCondition> ]
<tableReference>
        ::= <tableName> [AS <correlationName>]
<expansionSpec>
        ::= <tableName> [ ( <expandColumnList>]
                  [ WHERE <expansionPredicate>] ) ]
        ::= *
<expandoColumnList>]
        ::= <columnName> [ AS <columnName>]
                  [, <columnName> [ AS <columnName>]]*
        ::= *
<expansionPredicate>
        ::= <searchCondition>
<sortColumn>
        ::= [−] <columnSpecification>
```

It should be noted that all aspects of the above definition are the same as prior art SQL, except for the EXPAND, DEPTH and <expansionSpec> terms. An <expansionSpec> tells how to expand columns that are REFERENCEs to a specific table. In other words, it specifies which table and which reference columns in that table are to be used for expansion. The <expansionSpec> clause can also specify logical conditions which limit the rows that are retrieved during expansion.

Consider the effect of adding an EXPAND clause to the query discussed above for selecting the "ENGINE" row of the parts base table:

```
SELECT *
FROM parts
```

-continued

```
EXPAND parts(*)
WHERE partName = 'ENGINE'
```

Note that the "parts" table is the table 400 shown in FIG. 6. Like the prior art query, this query commands the retrieval of the row in table 400 for "ENGINE". Call this row the top-level row. The EXPAND clause

```
"EXPAND parts(*)"
```

directs that whenever a REFERENCE to the table called parts is encountered in any retrieved row, the reference is to be followed according to the expansion process described above. Thus, referring to FIG. 6, not only is the row for "ENGINE" retrieved, but also the ten rows with partName's of CAM SHAFT, CYLINDER 1, CYLINDER 2, CYLINDER 3, CYLINDER 4, CYLINDER 5, CYLINDER 6, CYLINDER 7, CYLINDER 8, and WATER JACKET.

In addition, since these rows also have a containsParts column with references to other rows in the table Parts, the expansion process continues with these rows, and so on. Specifically, each of the eight cylinders rows have references to an exhaust value, an intake valve, a piston, and a spark plug. Together, the rows for the eight cylinders reference thirty-two more part rows that are also retrieved, namely the rows for the parts:

EXHAUST VALVE 1	INTAKE VALVE 1	PISTON 1	SPARK PLUG 1
EXHAUST VALVE 2	INTAKE VALVE 2	PISTON 2	SPARK PLUG 2
...			
EXHAUST VALVE 8	INTAKE VALVE 8	PISTON 8	SPARK PLUG 8

Thus, the above query with the EXPAND specification causes the top-level row and forty-two additional rows reached through the top-level row to be retrieved, or forty-three retrieved rows in total. The additional forty-two rows retrieved by this query are structurally related to the top-level row through a path of references from the top-level row ("ENGINE") to each of the other rows retrieved.

Without the optional EXPAND clause, data stored in the columns specified in the <selectList> of a SELECT command are copied into a "top-level" record in the application program. Retrieved data from columns with a data type of REFERENCE is stored in a field or fields of the application's "top-level" record(s) that contain the foreign key values associated with the reference.

The optional EXPAND clause in a SELECT statement causes the specified REFERENCE columns to be expanded. A column of type REFERENCE is said to be expanded if it is retrieved into the application program as a pointer to another record, rather than as the foreign key values associated with the reference. The columns of the expanded row themselves become candidates for further expansion.

Only those columns that are of type REFERENCE are expanded. If the <expandColumnList> in the EXPAND clause is "*", then all columns of the table specified by the EXPAND clause are expanded If the <expansionSpec> in the EXPAND clause is "*", then all columns of all tables not mentioned in an expansion clause are expanded in their entirety, limited only by the DEPTH specified, if any.

Specifically, consider a column "c" whose type is REFERENCE and that refers to a particular row "r" in table "t". Column "c" will be expanded if all of the following conditions are met:

(i) an EXPAND clause is present and the EXPAND clause specifies that references to table "t" are to be expanded;

(ii) the "WHERE" condition associated with the particular <expansionSpec> used is TRUE for row "r", or there is no such "WHERE" condition; and

(iii) the number of previously expanded rows required to reach row "r" does not exceed the "DEPTH" number given in the "EXPAND" clause, or there is no DEPTH limit.

The columns of the row "r" that are retrieved and that become possibly candidates for expansion themselves are limited to the columns that appear in the <expandColumnList> following <tableName> "t" in the expansion clause. If the <expandColumnList> associated with table "t" is "*", then all columns of table "t" are retrieved and are candidates for expansion. Similarly, if the <expansionSpec> "*" appears, and <tableName> "t" does not occur in the EXPAND clause, then all columns of table "t" are retrieved and become candidates for expansion.

The candidate columns for expansion initially come from the columns whose names appear in <selectList> of the SELECT command. Once a column "c" that refers to row "r" is expanded, any columns of "r" that meet conditions (i), (ii), and (iii) above are expanded and become candidates for expansion themselves, at the next higher depth level.

The optional DEPTH clause limits the expansion to <n> levels. Specifying a depth of one (1) limits the retrieved data structure to one or more "top-level" records, which directly meet the criteria of the "WHERE" clause in the SELECT statement, with no other records below the top-level. A depth of three limits the expansion to three levels, including the top-level row and up to two levels below it. Note that the retrieved records, regardless of the specified DEPTH, will contain only those fields which correspond to the columns in <selectList> of the SELECT statement.

In the preferred embodiment, the depth number applies specifically to a "breadth first" expansion process. Thus, the retrieved directed graph data structure may, in general, contain cycles and/or multiple pointers to the same substructure, and thus there may be paths from the top-level record of the structure to other records in the structure that exceed the specified depth. In other words, if the specified DEPTH is equal to a positive integer n, and if the minimum number of references required to get from a top-level row to a particular second row is less than n, then that second row is retrieved even if there are other paths between the top-level row and the second row which traverse n or more reference pointers.

During a reference expansion, if a reference is incomplete (contains at least one NULL key value) or references a non-existent row, the reference is returned to the application as a NULL pointer. Similarly, if the DEPTH is exceeded, the reference is returned as a NULL pointer.

The data structures that may be retrieved as a result of a SELECT with an EXPAND clause include n-ary trees (n=0, 1, 2,...), linked lists, and, in general, any directed graph data structure. If the data in the database refer to the same row more than once, only a single instance of the row is retrieved Thus, only a single copy of each unique database row will be retrieved and stored as a record in the directed graph data structure that is selected.

Referring to FIGS. 3, 7 and 8, the procedure used by the DMBS to perform an expanded SELECT query and to store the retrieved data in the intermediate data format shown in FIG. 7, denoted in pseudocode form, is as follows.

PSEUDOCODE FOR RETRIEVAL WITH TRANSITIVE CLOSURE

```
INITIALIZATION:
    CLEAR HASH TABLE
    CLEAR LIST RROWS FOR STORING RETRIEVED ROWS
    CLEAR LIST OF TABLE DEFINITIONS TDEF
RETRIEVE TOP LEVEL ROWS IN ACCORDANCE WITH UNEXPANDED
SELECT STATEMENT
STORE TABLE DEFINITION FOR TOP LEVEL ROWS IN TDEF
STORED TOP LEVEL ROW(S) IN RROWS
STORE CORRESPONDING ENTRY OR ENTRIES IN HASH TABLE
FOR EACH ROW IN RROWS (BEGINNING WITH TOP LEVEL ROW)
    FOR EACH REFERENCE (IN ROW) MENTIONED IN EXPAND CLAUSE
        RR = RECORD POINTED TO BY REFERENCE
        IF RR IS NOT ALREADY IN HASH TABLE
          AND
```

-continued

```
        RR MEETS SPECIFIED WHERE <searchCondition> AND
        AND
        DEPTH OF RR FROM TOP LEVEL ROWS DOES NOT
        EXCEED DEPTH LIMIT
          ADD SPECIFIED COLUMNS OF RR, INCLUDING
          CALCULATED VALUES, TO RROWS
          IF TABLE DEFINITION FOR RR IS NOT ALREADY IN
          TDEF
              ADD TABLE DEFINITION FOR RR TO TDEF
          ENDIF
          ADD RR TO HAS TABLE
      ENDIF
    ENDLOOP
    USE LINK IN HASH TABLE TO FIND NEXT ROW, IF ANY
ENDLOOP
STORE TDEF AND RROWS IN BUFFER FORMAT SHOWN IN FIG. 7
TRANSMIT DATA IN BUFFER FORMAT TO APPLICATION INTERFACE
```

The initialization step in the above pseudocode routine sets up an empty hash table and two lists: one for retrieved rows called RROWS and one for table definitions called TDEF. Table definitions for all the distinct types of rows that are retrieved (i.e., from distinct base tables) are added to the TDEF list as the list of retrieved rows is accumulated.

The list of rows in the DBMS application task buffer 241 is constructed by visiting each retrieved row in the RROWS list once, and expanding those reference columns denoted in the EXPAND clause. Notice that each expansion of a row may add new rows to the end of RROWS. When the last row of RROWS is expanded, the transitive closure has been computed. The expansion process must eventually reach the end of RROWS because duplicate rows are not added to the RROWS list, and there are a finite number of rows in any database.

To prevent storing a record more than once, which would be the case when the directed graph contains cycles or multiple pointers to one base table row, the DBMS checks, upon visiting each potentially new row, whether that row is already in the RROWS list (by looking for the row in the hash table 490), and if so, it does not process that row any further. If the row has not already been visited, the DBMS checks the specified <expansionPredicate> (which is a type of search condition) to determine whether to add this row to the RROWS list. If so, the columns of the row specified by <selectList> are stored as a new row in the RROWS list. The row is also added to the hash table. Then the buffer building process moves onto the next row in the RROWS list, which is found using the link field of the hash table.

When all the rows in RROWS have been visited, the resulting lists TDEF and RROWS are stored in a buffer 241 by the DBMS in the format shown in FIG. 7. This retrieved data is logically equivalent to one or more derived tables. A derived table is a subset or portion of a base table from which data is being retrieved. The retrieved, buffered data is transmitted to the application interface 232, which then converts the retrieved list of rows into a directed graph data structure suitable for use by the application program which generated the retrieval query.

Referring to FIG. 7, the procedure for converting the retrieved data (stored in buffer 242 in the intermediate data format shown in FIG. 7) into a directed graph data structure, denoted in pseudocode form, is as follows.

PSEUDOCODE FOR CONVERTING RETRIEVED DATA INTO DIRECTED GRAPH

```
READ RETRIEVED DATA IN APPLICATION INTERFACE BUFFER
ALLOCATE AN ARRAY OF POINTERS REFROWS TO STORE POINTERS TO
EACH OF THE ROWS IN THE BUFFER
FOR EACH ROW R IN THE BUFFER
    ALLOCATE A RECORD THAT CONTAINS FIELDS FOR EACH OF THE
    RETRIEVED COLUMNS OF DATA, AS WELL AS ANY ADDITIONAL
    FIELDS SPECIFIED BY THE APPLICATION PROGRAM
    (RETRIEVED DATA IS INTERPRETED USING THE CORRESPONDING
    TABLE DEFINITION IN THE BUFFER)
    STORE POINTER TO ALLOCATED RECORD IN CORRESPONDING
    POSITION OF REFROWS
ENDLOOP
FOR EACH ROW R IN THE BUFFER
    STORE DATA FOR THE ROW R FROM THE BUFFER INTO THE
    ALLOCATED RECORD
    FOR EACH FIELD IN THE ALLOCATED RECORD WHICH
    CORRESPONDS TO AN EXPANSION COLUMN
        REPLACE ROW NUMBER (N) WITH POINTER IN REFROWS (N)
    ENDLOOP
ENDLOOP
ARG = FIRST POINTER IN REFROWS -   Pointer to Top Level
                                   Record of Directed Graph
DEALLOCATE REFROWS ARRAY
PASS ARG TO APPLICATION PROGRAM
```

When a SELECT query with an EXPAND clause is specified, the application interface uses the declarative information about the tables involved, stored within the table definition portion of the buffer that it receives from the DBMS, to transform the "flat" buffer representation into a directed graph data structure in the application programming language's representation.

Each application programming language has its own data structure representation mechanisms and its own data types. Thus, the representation of retrieved directed graph data structures differs somewhat from one computer programming language to another.

In the preferred embodiment, when a SELECT query with an EXPAND clause is issued, the application interface accepts from the application program some optional record descriptions, in conjunction with the query itself, that enables the application program to specify the exact memory layout to use for each table whose rows are being retrieved as part of the directed graph data structure.

The form in which record descriptions are provided is similar to the way records are described when a USING ARG query is issued, as described above. During the translation of retrieved data (stored in the intermediate buffer format) into a directed graph data structure, the application interface uses the record descriptions for each type of record being retrieved to decide how to allocate the record and where to store each field.

Allowing the application to provide this optional information has two significant advantages. First, it ensures that a record template can be declared in the application programming language to match the records constructed by the interface. Thus convenient language constructs can be used to access records. Second, it allows the application program to include in each record some additional fields that do not correspond to retrieved data, but that the program may need to perform certain operations on the records. An example of an additional field is a flag field that the application program can set during its own processing to mark that a particular record should be modified or deleted in the database. This field may then be referred to in the <searchCondition> associated with a later DELETE ... USING ARG query, for example.

In summary, the directed graph data structure is generated by storing each row in the buffer in the form of a record in the format expected by the application program. Furthermore, expanded references which are represented in the intermediate buffer format as row numbers are replaced with pointers to the corresponding records in the converted data structure. The resulting data structure is then passed by the application interface 232 to the application program 230.

CONCLUSIONS AND ALTERNATE EMBODIMENTS

It should be noted that the processing of each query by a DBMS requires a certain amount of system resources to process the query, regardless of how simple or complex the query command may be. Thus it takes considerably less system resources for a DBMS to process a single extended query than it takes to process numerous prior art queries, even though the single query retrieves the same amount of data as the numerous prior art queries.

In various uses of the present invention, the number of prior art queries replaced by single extended query will depend on the complexity of the data structures being used and the amount of data that needs to be retrieved. In some contexts, a single extended query may replace hundreds of prior art queries. In addition, the application interface feature of the present invention

automatically converts directed graph data structures into table form, and vice versa, thereby reducing the complexity of engineering application programs which use a DBMS to store directed graph data structures.

While the present invention has been described with reference to a few specific embodiments, the description is illustrative of the invention and is not to be construed as limiting the invention. Various modifications may occur to those skilled in the art without departing from the true spirit and scope of the invention as defined by the appended claims.

What is claimed is:

1. A computer system for storing, retrieving and modifying data stored in a database, comprising:
a database server; and
a multiplicity of application processes coupled to said database server, each application process including:
an application program that utilizes directed graph data structures in a corresponding application specific data format; each said directed graph data structure including one or more records of data interconnected by pointers, each record composed of one or more data elements having respective specified data types;
an application interface that translates directed graph data structures in said application specific data format into a predefined intermediate data format and translates directed graph data structures in said predefined intermediate data a format into said application specific data format; and
query generating means for generating and sending queries to said database server for storing, retrieving and updating specified directed graph data structures in said database;
said database server including:
schema defining means for defining a distinct schema for each of a plurality of tables in said database, each said database table having a plurality of rows and a specified number of columns, wherein each row of said table stores data values in each of said columns of said table; said schema denoting a data type for each column of data values stored in said table, each denoted data type being selected from a set of predefined data types including a reference data type;
each non-empty data value stored in a reference data type column comprising a reference to a row of one of said tables in said database; and
directed graph storage means for storing and retrieving specified directed graph data structures in and from specified tables in said database in accordance with queries received from said application processes;
said directed graph storage means including:
graph storing means for receiving a data storage query that includes a specified directed graph data structure in said predefined intermediate data format and for storing each record of said received directed graph data structure in a distinct row of a respective one of said tables in said database, said graph storing means including means for storing said data elements of said each record in corresponding columns of said one respective table and means for storing references, each reference corresponding to one of said pointers that interconnect said each record

to other records of said specified directed graph data structure, in corresponding ones of said reference data type columns of said one respective table; and

directed graph retrieving means for retrieving a specified directed graph data structure from said database in accordance with a single specified query received from one of said application processes, including means for retrieving in accordance with said specified query at least one specified row from at least one respective table in said database and then retrieving additional rows of data from respective tables in said database, said additional rows of data comprising rows of data that are referenced by references in other rows of data retrieved in accordance with said specified query, for converting said retrieved rows of data into a directed graph data structure in said predefined intermediate data format, and for transmitting said retrieved directed graph data structure to said one application process;

whereby a directed graph data structure having multiple records is retrieved by said database server in response to a single query from said one application process.

2. A computer system as in claim 1,

said schema defining means denoting in each said schema a table identifier for each reference data type column, said table identifier specifying which database table is referenced by references in that column;

each said table in said database having an associated primary key comprising an ordered list of columns of said table whose values identify a particular row of said table; and

each of said references comprising a primary key value of a row in the respective database table specified by the schema for the table in which each said reference is stored.

3. A computer system as in claim 1, wherein each directed graph data structure in said intermediate data format includes rows of data values, each row of data values corresponding to a respective record of a corresponding directed graph data structure in one said application specific data format, said rows of data values including row pointers interconnecting said rows of data values, each row pointer corresponding to a respective pointer in said corresponding directed graph data structure in one said application specific data format.

4. A computer system as in claim 1,

said query generating means including means for including in said generated queries specified criteria for limiting retrieval of said additional rows of data.

5. A computer system as in claim 1,

said query generating means including means for including retrieval limiting criteria in ones of said generated queries, said retrieval limiting criteria denoting a maximum depth, said maximum depth comprising a maximum number of pointers used in sequence to interconnect one or more top-level records of said specified directed graph data structure with other records of said specified directed graph data structure; and

said directed graph retrieving means including means for limiting retrieval of said additional rows of data, when retrieving a specified directed graph

data structure in accordance with a query including retrieval limiting criteria, to those of said additional rows of data that are connected to said specified at least one row of data by a sequence of references no greater in number than said maximum number denoted by said retrieval limiting criteria.

6. A computer system for storing, retrieving and modifying data stored in a database, comprising:

a database server;

a multiplicity of application processes coupled to said database server, each application process including:

an application program that utilizes directed graph data structures in a corresponding application specific data format; each said directed graph data structure including one or more records of data interconnected by pointers, each record composed of one or more data elements having respective specified data types;

query generating means for sending queries to said database server for storing, retrieving said updating specified directed graph structures in said database; and

at least one application interface that translates directed graph data structures in one respective application specific data format into a predefined intermediate data format and translates directed graph data structures in said predefined intermediate data format into said one respective application specific data format; said at least one application interface coupling said database server to a respective at least one of said application processes;

said database server including:

schema defining means for defining a distinct schema for each of a plurality of tables in said database, each said database table having a plurality of rows and a specified number of columns, wherein each row of said table stores data values in each of said columns of said table; said schema denoting a data type for each column of data values stored in said table, each denoted data type being selected from a set of predefined data types including a reference data type;

each non-empty data value stored in a reference data type column comprising a reference to a row of one of said tables in said database; and

directed graph storage means for storing and retrieving specified directed graph data structures in and from specified tables in said database in accordance with queries received from said application processes;

said directed graph storage means including:

graph storing means for receiving a data storage query that includes a specified directed graph data structure in said predefined intermediate data format and for storing each record of said received directed graph data structure in a distinct row of a respective one of said tables in said database, said graph storing means including means for storing said data elements of said each record in corresponding columns of said one respective table and means for storing references, each reference corresponding to one of said pointers that interconnect said each record to other records of said specified directed graph data structure, in corresponding ones of said reference data type columns of said one respective table; and directed graph retrieving means for retrieving a specified directed graph from said database in accordance with a single specified query received from one of said application processes, including means for retrieving in accordance with said specified query at least one specified row from at least one respective table in said database and then retrieving additional rows of data from respective tables in said database, said additional rows of data comprising rows of data that are referenced by references in other rows of data retrieved in accordance with said specified query, for converting said retrieved rows of data into a directed graph data structure in said predefined intermediate data a format, and for transmitting said retrieved directed graph data structure to said one application process;

whereby a directed graph having multiple records is retrieved by said database server in response to a signal query from said one application process.

7. A computer system as in claim 6,

said schema defining means denoting in each said schema a table identifier for each reference data type column, said table identifier specifying which database table is referenced by references in that column;

each said table in said database having an associated primary key comprising an ordered list of columns of said table whose values identify a particular row of said table; and

each of said references comprising a primary key value of a row in the respective database table specified by the schema for the table in which each said reference is stored.

8. A computer system as in claim 6, wherein each directed graph data structure in said intermediate data format includes rows of data values, each row of data values corresponding to a respective record of a corresponding directed graph data structure in one said application specific data format, said rows of data values including row pointers interconnecting said rows of data values, each row pointer corresponding to a respective pointer in said corresponding directed graph data structure in one said application specific data format.

9. A computer system as in claim 6,

said query generating means including means for including in said generated queries specified criteria for limiting retrieval of said additional rows of data.

10. A computer system as in claim 6,

said query generating means including means for including retrieval limiting criteria in ones of said generated queries, said retrieval limiting criteria denoting a maximum depth, said maximum depth comprising a maximum number of pointers used in sequence to interconnect one or more top-level records of said specified directed graph data structure with other records of said specified directed graph data structure; and

said directed graph retrieving means including means for limiting retrieval of said additional rows of data, when retrieving a specified directed graph data structure in accordance with a query including retrieval limiting criteria, to those of said additional rows of data that are connected to said specified at least one row of data by a sequence of references no greater in number than said maximum number denoted by said retrieval limiting criteria.

11. In a computer system, a control process of storing and retrieving directed graph data structures in a data table in a computer system; said computer system having a multiplicity of application processes coupled to a database server that responds to queries from said application processes by storing, retrieving and updating data in said database; the steps of the control process comprising:

in each application process, executing an application program that utilizes directed graph data structures in a corresponding application specified data format; each said directed graph data structure including one or more records of data interconnected by pointers, each record composed of one or more data elements having respective specified data types;

each application process generating and sending queries to said database reserver for storing, retrieving and updating specified directed graph structures in said database;

when transmitting directed graph data structures from any one of said application processes to said database server, translating said transmitted directed graph data structures from the application specific data format utilized by said one of said application processes into a predefined intermediate data format,

when transmitting directed graph data structures from said database server to any one of said application processes, translating said transmitted directed graph data structures from said predefined intermediate data format into the application specific data format utilized by said one of said application processes;

in said database server, defining a distinct schema for each of a plurality of tables in said database, each said database table having a plurality of rows and a specified number of columns, wherein each row of said database table stores data values in each of said columns of said database table; said schema denoting a data type for each column of data values stored in said database table, each denoted data type being selected from a set of predefined data types including a reference data type; type;

each non-empty data value stored in a reference data type column comprising a reference to a row of one of said tables in said database;

in said database server, storing and retrieving specified directed graph data structures in and from specified tables in said database in accordance with queries received from one of said application processes; said storing and retrieving steps including:

receiving from one of said application processes a data storage query that includes a specified directed graph data structure in said predefined intermediate data format, and storing each record of said received directed graph data structure in a distinct row of a respective one of said database tables, said record storing step including storing said data elements of said each record in corresponding columns of said one respective database table and storing references, each reference corresponding to one of said pointers that interconnect said each record to other records of said specified directed graph data structure, in corresponding ones of said reference data type columns of said one respective database table; and

retrieving a specified directed graph data structure from said database in accordance with a single specified query received from one of said application processes, including retrieving in accordance with said specified query at least one specified row from at least one respective database table and then retrieving additional rows of data from respective ones of said database tables, said additional rows of data comprising rows of data that are referenced by references in other rows of data retrieved in accordance with said specified query, converting said retrieved rows of data into a directed graph data structure in said predefined intermediate data format, and transmitting said retrieved directed graph data structure to said one application process.

12. The control process of claim 11,

said schema defining step including denoting in each said schema a table identifier for each reference data type column, said table identifier specifying which database table is referenced by references in that column;

each said table in said database having an associated primary key comprising an ordered list of columns of said table whose values identify a particular row of said table; and

each of said references comprising a primary key value of a row in the respective database table specified by the schema for the table in which each said reference is stored.

13. The control process of claim 11, wherein each directed graph data structure in said intermediate data format includes rows of data values, each row of data values corresponding to a respective record of a corresponding directed graph data structure in the application specific data format utilized by one of said application processes, said rows of data values including row pointers interconnecting said rows of data values, each row pointer corresponding to a respective pointer in said corresponding directed graph data structure in said application specific data format.

14. The control process of claim 11,

said query generating step including the step of including in ones of said generated queries specified criteria for limiting retrieval of said additional rows of data.

15. The control process of claim 11,

said query generating means including means for including retrieval limiting criteria in ones of said generated queries, said retrieval limiting criteria denoting a maximum depth, said maximum depth comprising a maximum number of pointers used in sequence to interconnect one or more top-level records of said specified directed graph data structure with other records of said specified directed graph data structure; and

said step of retrieving a specified directed graph data structure including limiting retrieval of said additional rows of data, when retrieving a specified directed graph data structure in accordance with a query including retrieval limiting criteria, to those of said additional rows of data that are connected to said specified at least one row of data by a sequence of references no greater in number than said maximum number denoted by said retrieval limiting criteria.

* * * * *

US005720023A

United States Patent [19]

Putland et al.

[11] **Patent Number: 5,720,023**

[45] **Date of Patent: Feb. 17, 1998**

[54] **APPARTUS AND METHOD FOR STORING DIAGRAM DATA**

[75] Inventors: **Paul Anthony Putland**, Ipswich; **Peter John Skevington**, Woodbridge; **Ian David Edmund Videlo**, Woodbridge; **John Peter Wittgreffe**, Woodbridge; **Martin John Yates**, Stowmarket, all of United Kingdom

[73] Assignee: **British Telecommunications public limited company**, London, United Kingdom

[21] Appl. No.: **647,922**

[22] PCT Filed: **Mar. 22, 1995**

[86] PCT No.: **PCT/GB95/00631**

§ 371 Date: **May 29, 1996**

§ 102(e) Date: **May 29, 1996**

[87] PCT Pub. No.: **WO95/26532**

PCT Pub. Date: **Oct. 5, 1995**

Related U.S. Application Data

[63] Continuation-in-part of Ser. No. 263,221, Jun. 21, 1994, abandoned.

[30] **Foreign Application Priority Data**

Mar. 28, 1994 [EP] European Pat. Off. 94302210

[51] **Int. Cl.**^{6} **G06F 15/00**

[52] **U.S. Cl.** **395/140**

[58] **Field of Search** 395/140, 141, 395/133; 345/117, 118

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,843,569 6/1989 Sawada et al. 364/518

5,617,523 4/1997 Imazu et al. 395/140

FOREIGN PATENT DOCUMENTS

0415796 A3 3/1991 European Pat. Off. .

OTHER PUBLICATIONS

Yamamoto et al, "Extraction of Object Features from Image and Its Application to Image Retriever", 9th International Conference on Pattern Recognition, 14–17 Nov. 1988, Rome, Italy, pp. 988–991.

Goers et al, "Definition and Application of Metaclasses in an Object–Oriented Database Model", 1993 IEEE, pp. 373–380, Department of Computer Science, Technical University of Clausthal, Germany.

Primary Examiner—Phu K. Nguyen
Attorney, Agent, or Firm—Nixon & Vanderhye P.C.

[57] **ABSTRACT**

An apparatus for storing data for use in displaying diagrams comprises a computer and a file server. Each diagram includes boxes and flow lines connecting the boxes. The software includes a database and a main program which is responsible for storing and retrieving diagram data as well as performing predetermined operations on the data. In the database, data describing a box is stored as an instance of a first software object class, data describing a flow line is stored as an instance of a second software object class, general data relating to a diagram is stored as an instance of a third software object class, and data describing the location of each entity (box or flow line) on a diagram is stored as an instance of a fourth software object class. Each instance of the fourth software object class includes a pointer to the instance of the third software object class which contains general data for the diagram and also to a pointer to the instance of the first or second software object class which describe the entity.

19 Claims, 20 Drawing Sheets

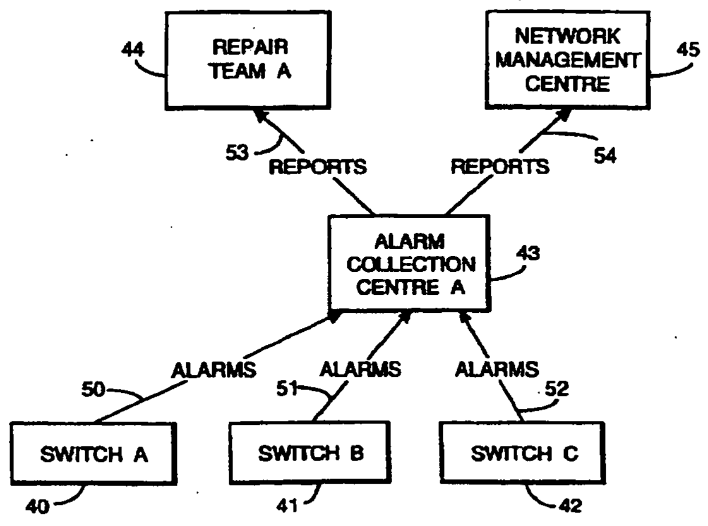


Fig.1.

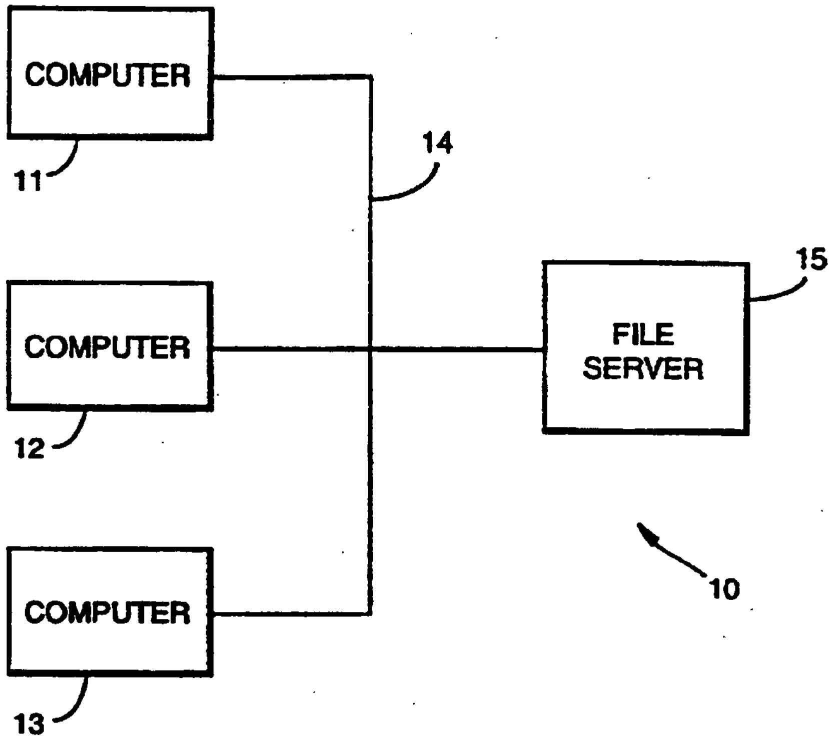

COMPUTER
11
COMPUTER
12
COMPUTER
13
14
FILE SERVER
15
10

Fig.2.

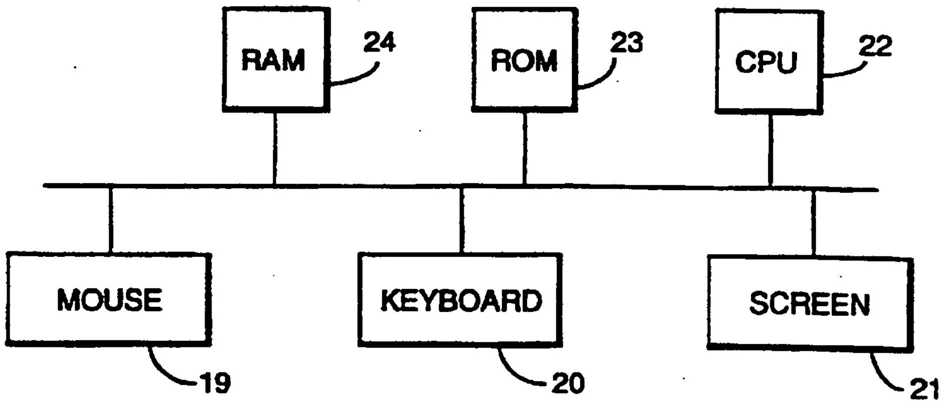

RAM
24
ROM
23
CPU
22
MOUSE
19
KEYBOARD
20
SCREEN
21

Fig.3.

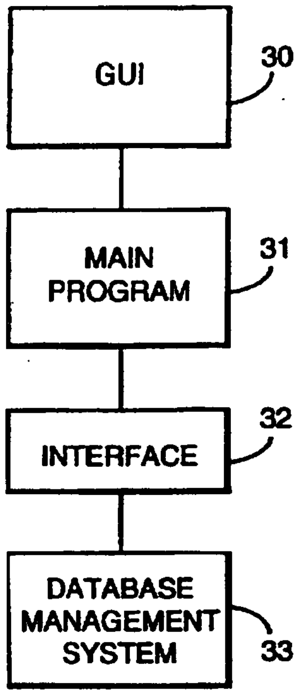

GUI
30
MAIN PROGRAM
31
INTERFACE
32
DATABASE MANAGEMENT SYSTEM
33

Fig.4.

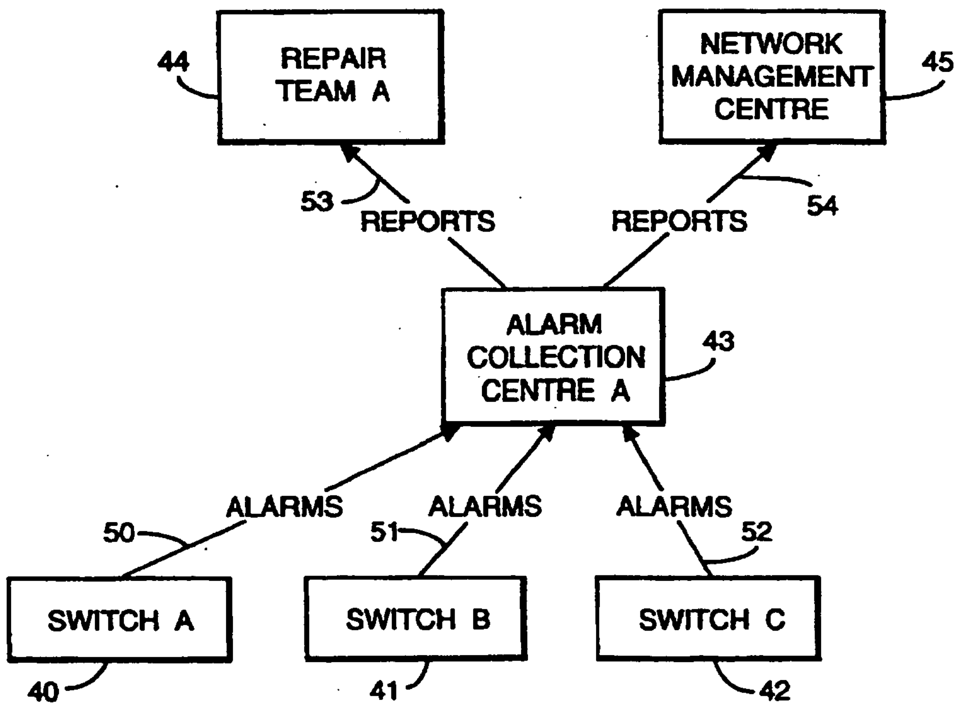

REPAIR TEAM A
44
NETWORK MANAGEMENT CENTRE
45
53
REPORTS
REPORTS
54
ALARM COLLECTION CENTRE A
43
ALARMS
ALARMS
ALARMS
50
51
52
SWITCH A
SWITCH B
SWITCH C
40
41
42

Fig.5.

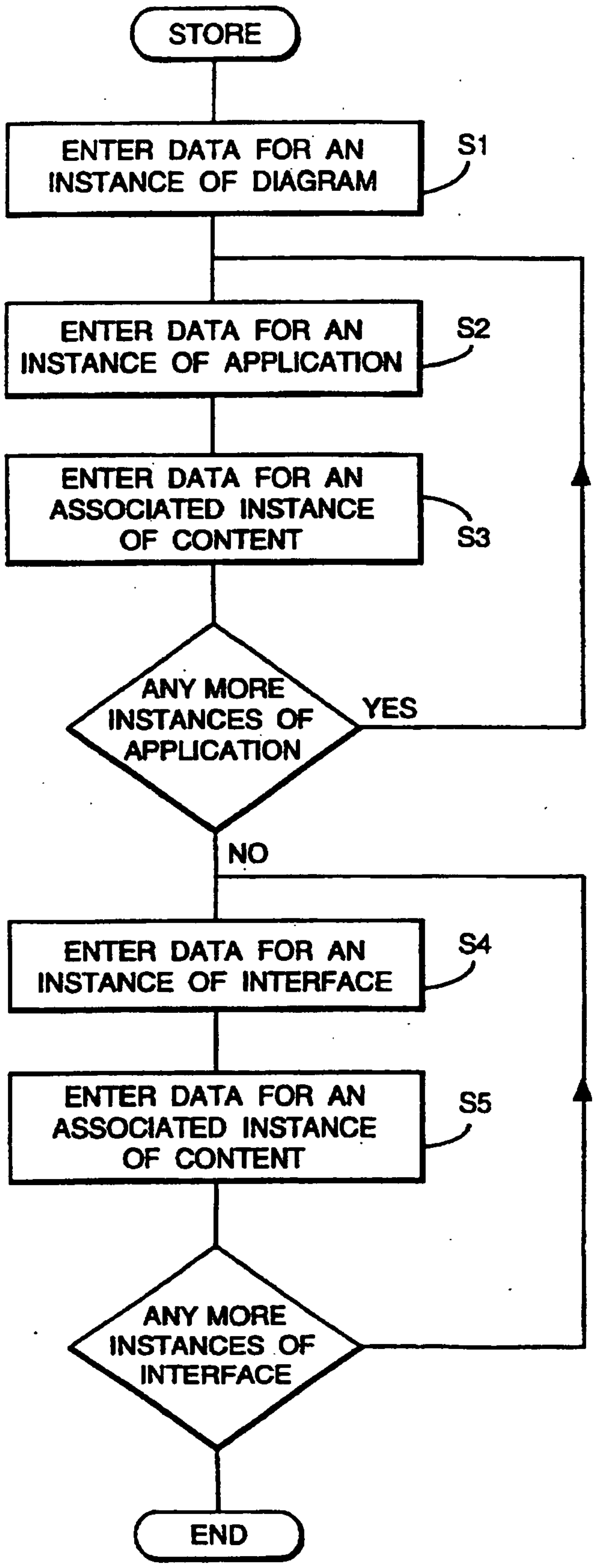

STORE
ENTER DATA FOR AN INSTANCE OF DIAGRAM
S1
ENTER DATA FOR AN INSTANCE OF APPLICATION
S2
ENTER DATA FOR AN ASSOCIATED INSTANCE OF CONTENT
S3
ANY MORE INSTANCES OF APPLICATION
YES
NO
ENTER DATA FOR AN INSTANCE OF INTERFACE
S4
ENTER DATA FOR AN ASSOCIATED INSTANCE OF CONTENT
S5
ANY MORE INSTANCES OF INTERFACE
END

Fig.6a.

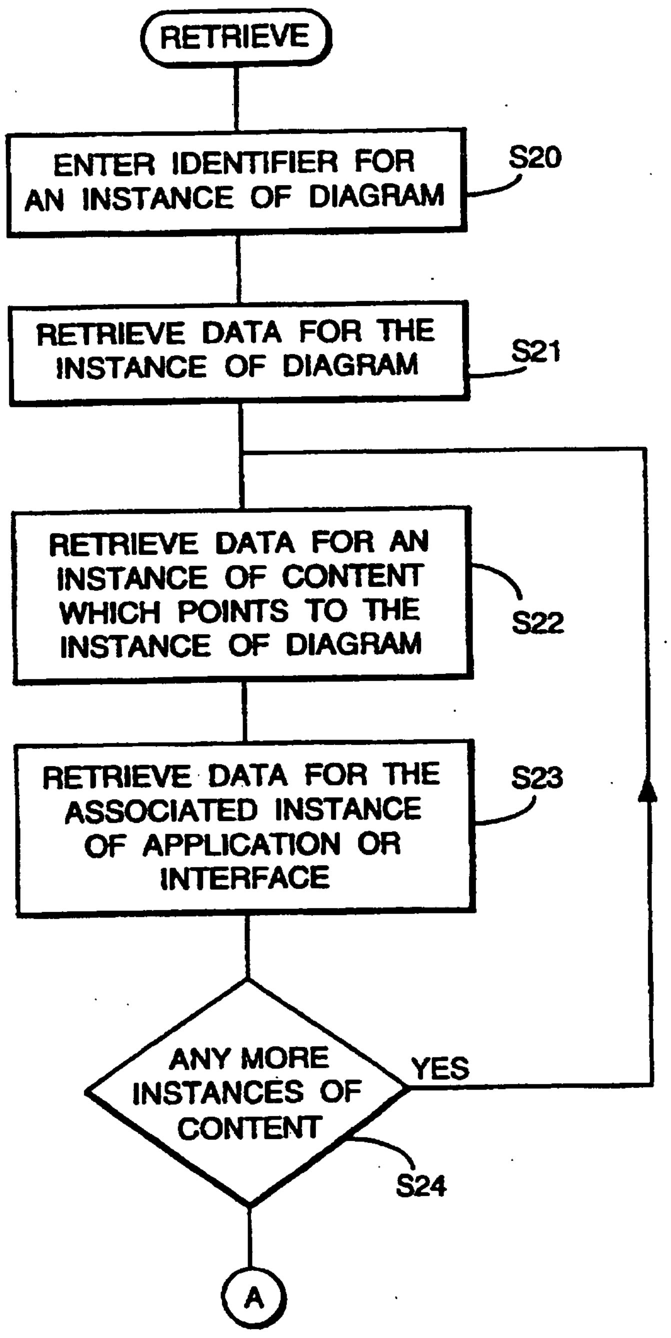

RETRIEVE
ENTER IDENTIFIER FOR AN INSTANCE OF DIAGRAM
S20
RETRIEVE DATA FOR THE INSTANCE OF DIAGRAM
S21
RETRIEVE DATA FOR AN INSTANCE OF CONTENT WHICH POINTS TO THE INSTANCE OF DIAGRAM
S22
RETRIEVE DATA FOR THE ASSOCIATED INSTANCE OF APPLICATION OR INTERFACE
S23
ANY MORE INSTANCES OF CONTENT
YES
S24
A

Fig.6b.

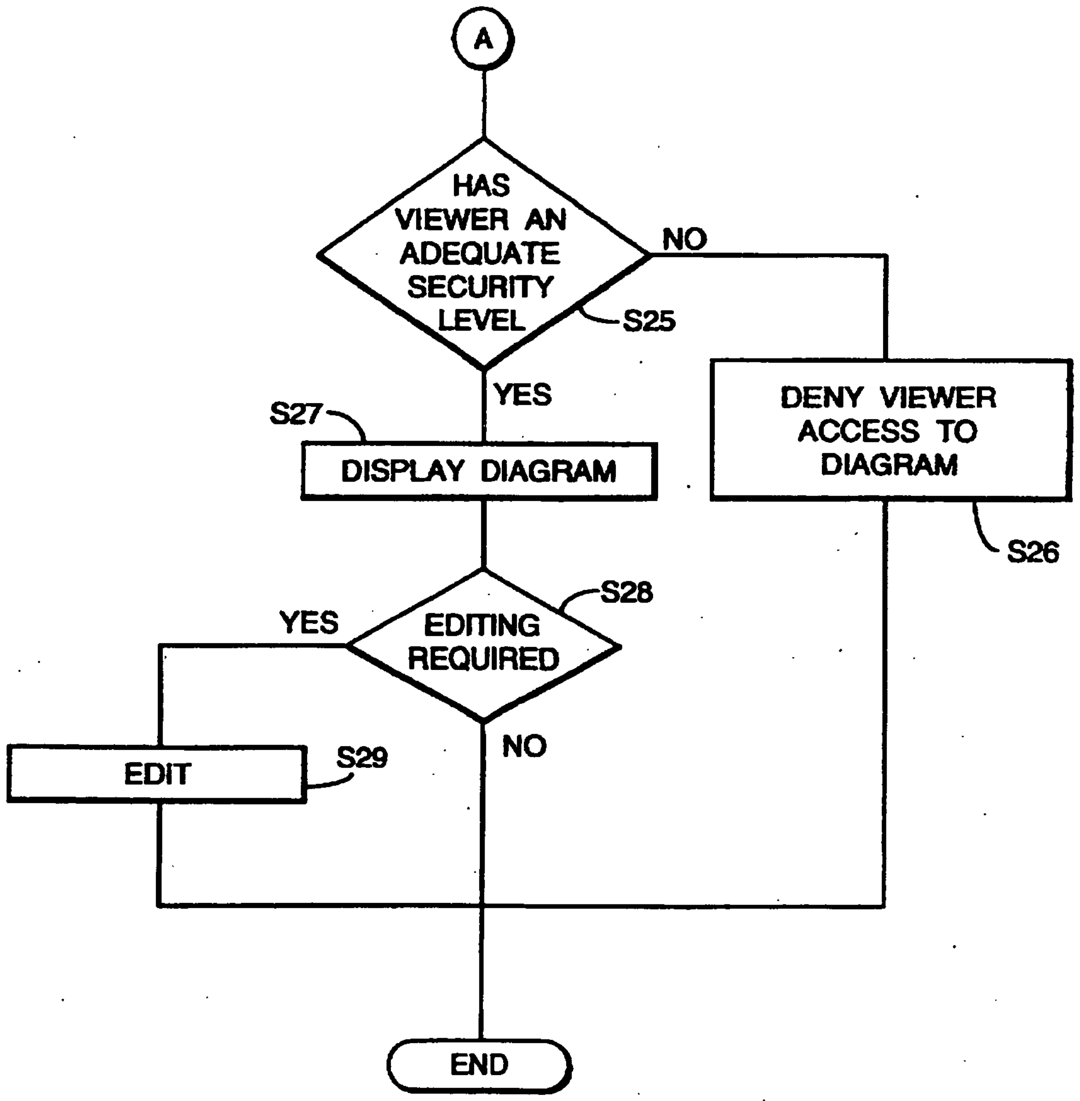

A
HAS VIEWER AN ADEQUATE SECURITY LEVEL
S25
NO
YES
DENY VIEWER ACCESS TO DIAGRAM
S26
S27
DISPLAY DIAGRAM
S28
EDITING REQUIRED
YES
NO
EDIT
S29
END

Fig.7.

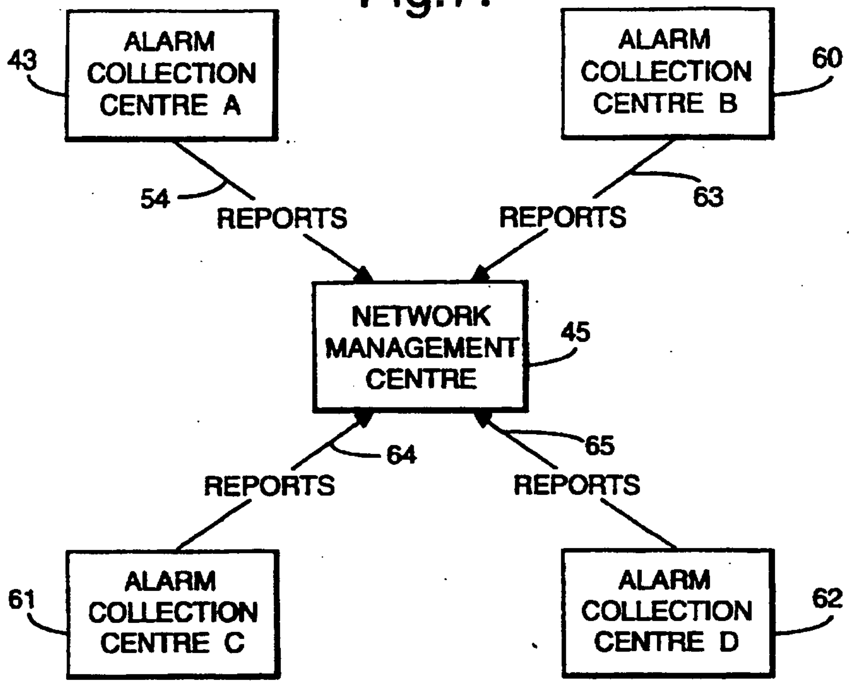

ALARM COLLECTION CENTRE A
43
ALARM COLLECTION CENTRE B
60
54
REPORTS
REPORTS
63
NETWORK MANAGEMENT CENTRE
45
64
65
REPORTS
REPORTS
ALARM COLLECTION CENTRE C
61
ALARM COLLECTION CENTRE D
62

Fig.12.

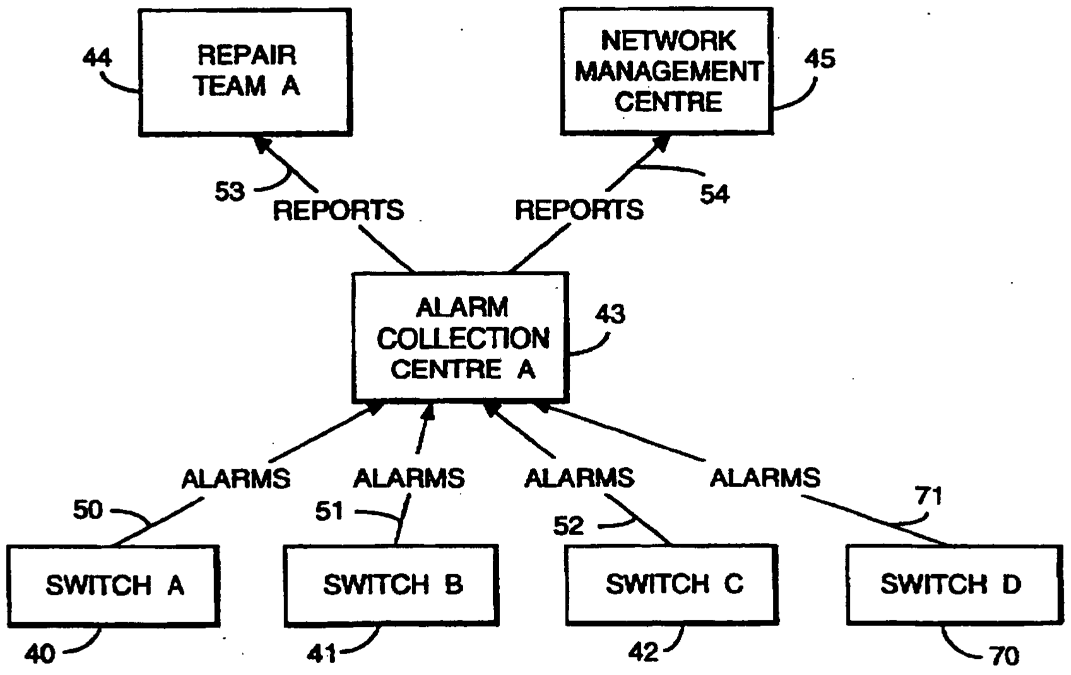

REPAIR TEAM A
44
NETWORK MANAGEMENT CENTRE
45
53
REPORTS
REPORTS
54
ALARM COLLECTION CENTRE A
43
ALARMS
ALARMS
ALARMS
ALARMS
50
51
52
71
SWITCH A
SWITCH B
SWITCH C
SWITCH D
40
41
42
70

Fig.8a.

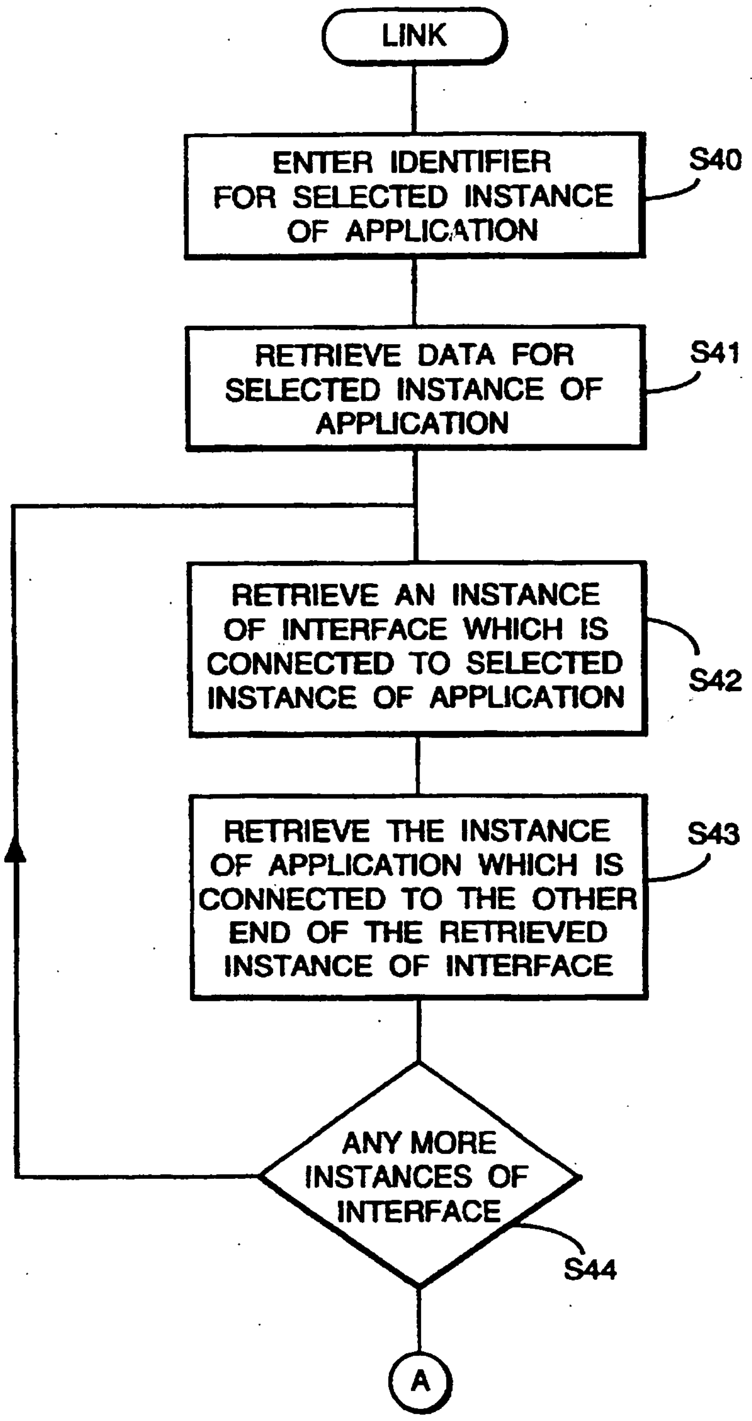

LINK
ENTER IDENTIFIER FOR SELECTED INSTANCE OF APPLICATION
S40
RETRIEVE DATA FOR SELECTED INSTANCE OF APPLICATION
S41
RETRIEVE AN INSTANCE OF INTERFACE WHICH IS CONNECTED TO SELECTED INSTANCE OF APPLICATION
S42
RETRIEVE THE INSTANCE OF APPLICATION WHICH IS CONNECTED TO THE OTHER END OF THE RETRIEVED INSTANCE OF INTERFACE
S43
ANY MORE INSTANCES OF INTERFACE
S44
A

Fig.8b.

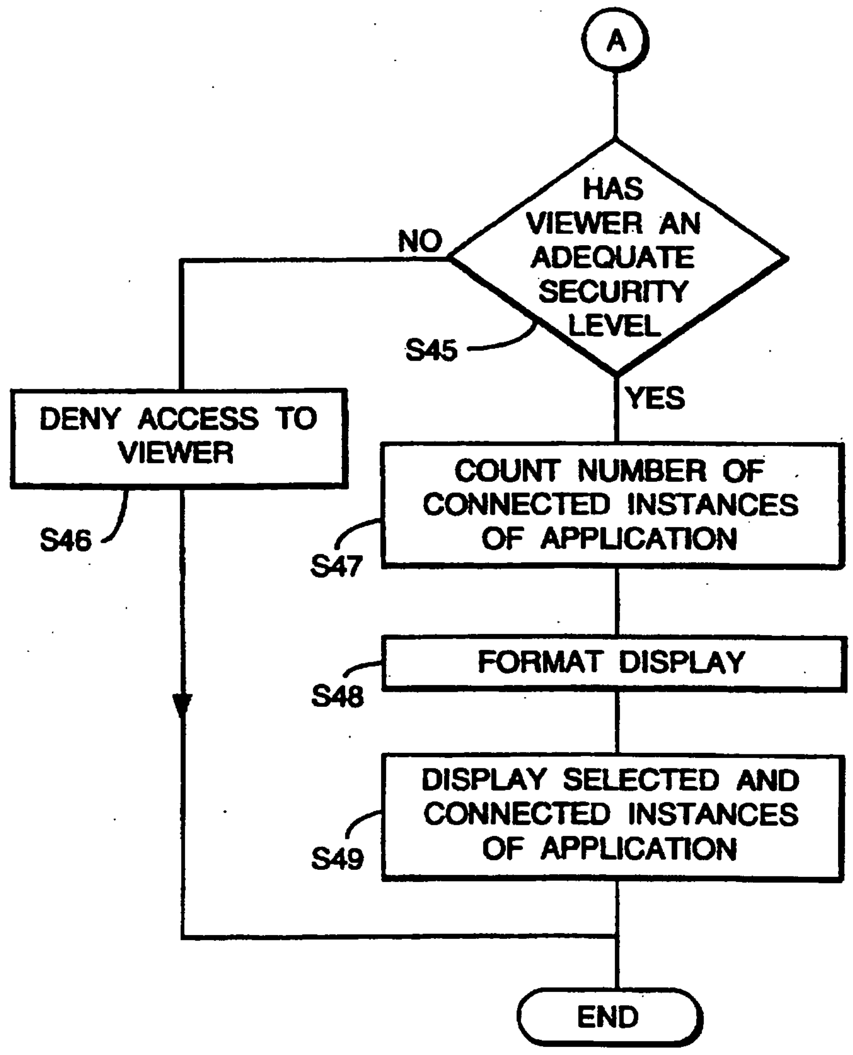
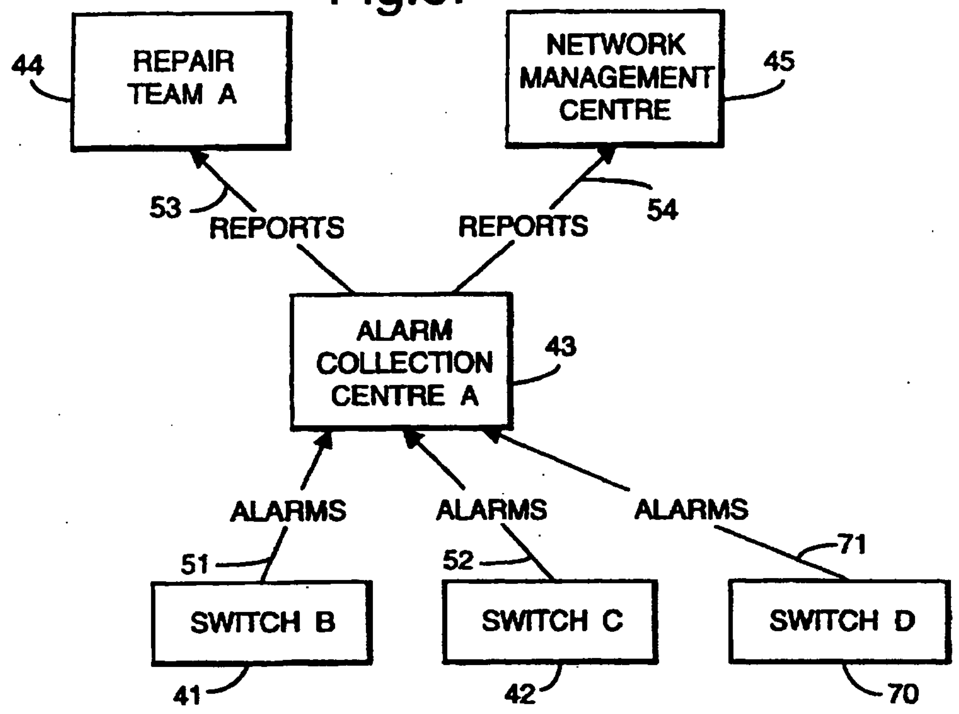
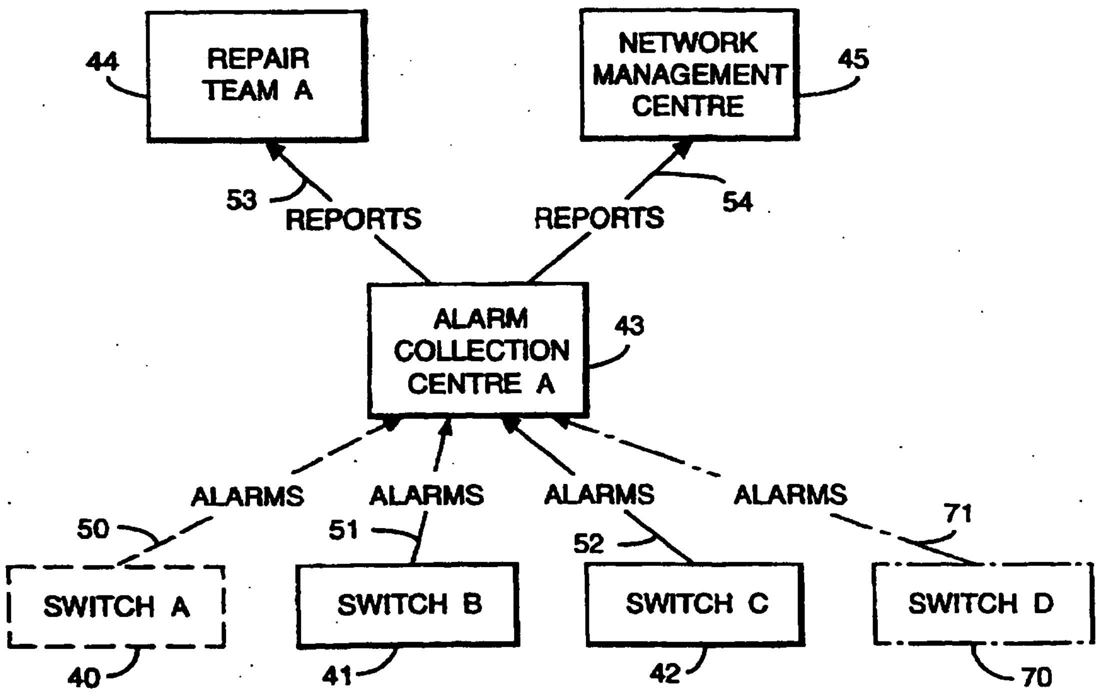

A
HAS VIEWER AN ADEQUATE SECURITY LEVEL
NO
S45
YES
DENY ACCESS TO VIEWER
S46
COUNT NUMBER OF CONNECTED INSTANCES OF APPLICATION
S47
FORMAT DISPLAY
S48
DISPLAY SELECTED AND CONNECTED INSTANCES OF APPLICATION
S49
END

Fig.9.



REPAIR TEAM A
44
NETWORK MANAGEMENT CENTRE
45
53
REPORTS
REPORTS
54
ALARM COLLECTION CENTRE A
43
ALARMS
ALARMS
ALARMS
51
52
71
SWITCH B
SWITCH C
SWITCH D
41
42
70

Fig.10.



REPAIR TEAM A
44
NETWORK MANAGEMENT CENTRE
45
53
REPORTS
REPORTS
54
ALARM COLLECTION CENTRE A
43
ALARMS
ALARMS
ALARMS
ALARMS
50
51
52
71
SWITCH A
SWITCH B
SWITCH C
SWITCH D
40
41
42
70

Fig.11a.

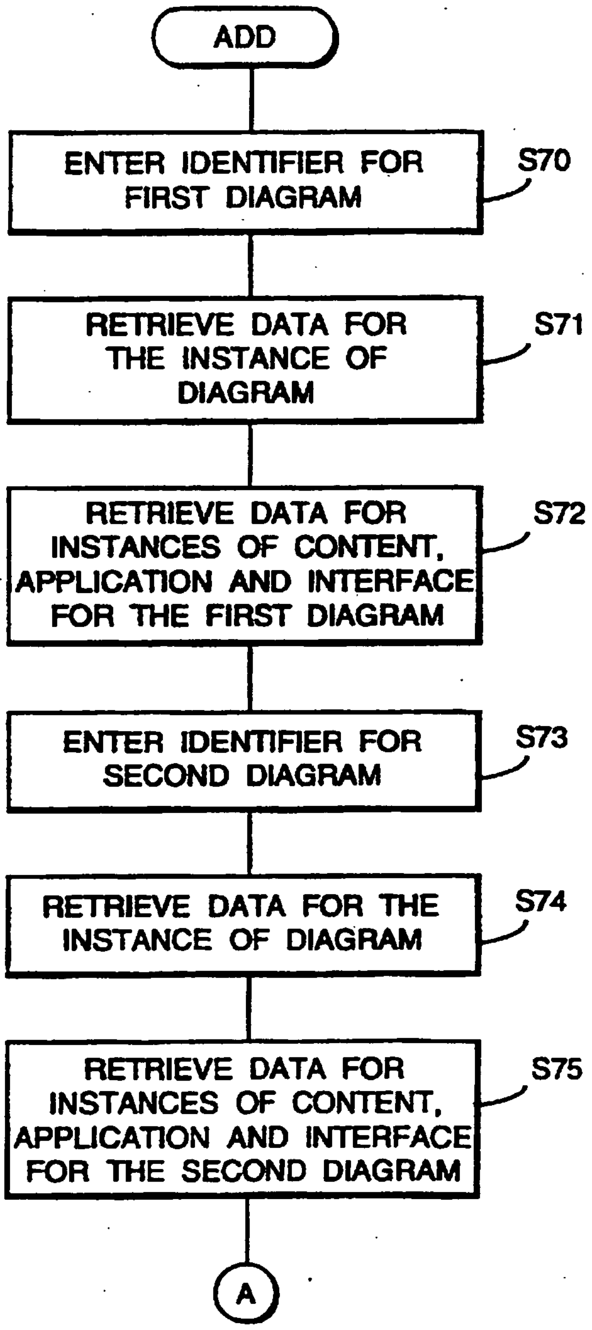

ADD
ENTER IDENTIFIER FOR FIRST DIAGRAM
S70
RETRIEVE DATA FOR THE INSTANCE OF DIAGRAM
S71
RETRIEVE DATA FOR INSTANCES OF CONTENT, APPLICATION AND INTERFACE FOR THE FIRST DIAGRAM
S72
ENTER IDENTIFIER FOR SECOND DIAGRAM
S73
RETRIEVE DATA FOR THE INSTANCE OF DIAGRAM
S74
RETRIEVE DATA FOR INSTANCES OF CONTENT, APPLICATION AND INTERFACE FOR THE SECOND DIAGRAM
S75
A

Fig.11b.

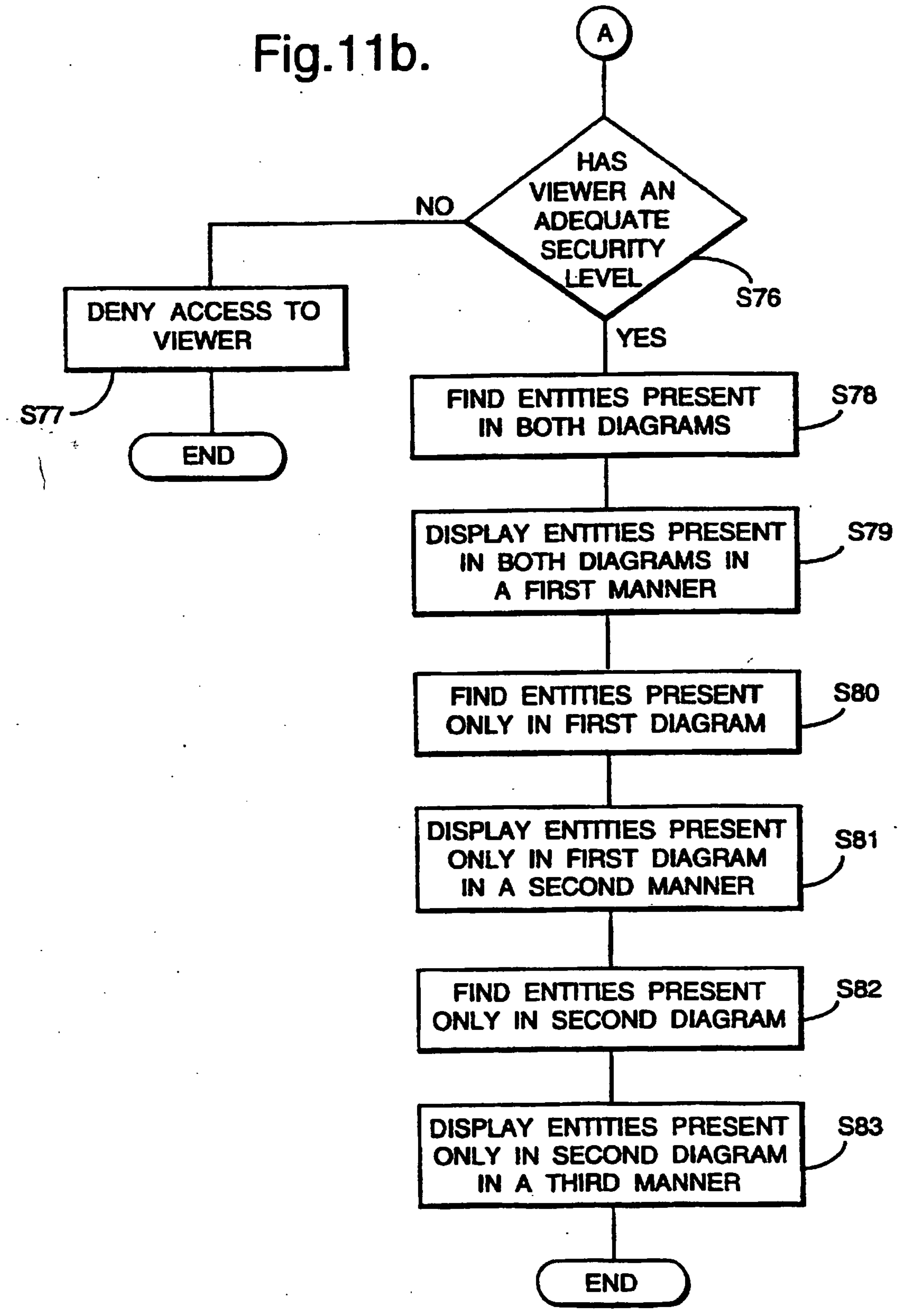

A
HAS VIEWER AN ADEQUATE SECURITY LEVEL
NO
YES
S76
DENY ACCESS TO VIEWER
S77
END
FIND ENTITIES PRESENT IN BOTH DIAGRAMS
S78
DISPLAY ENTITIES PRESENT IN BOTH DIAGRAMS IN A FIRST MANNER
S79
FIND ENTITIES PRESENT ONLY IN FIRST DIAGRAM
S80
DISPLAY ENTITIES PRESENT ONLY IN FIRST DIAGRAM IN A SECOND MANNER
S81
FIND ENTITIES PRESENT ONLY IN SECOND DIAGRAM
S82
DISPLAY ENTITIES PRESENT ONLY IN SECOND DIAGRAM IN A THIRD MANNER
S83
END

Fig.13a.

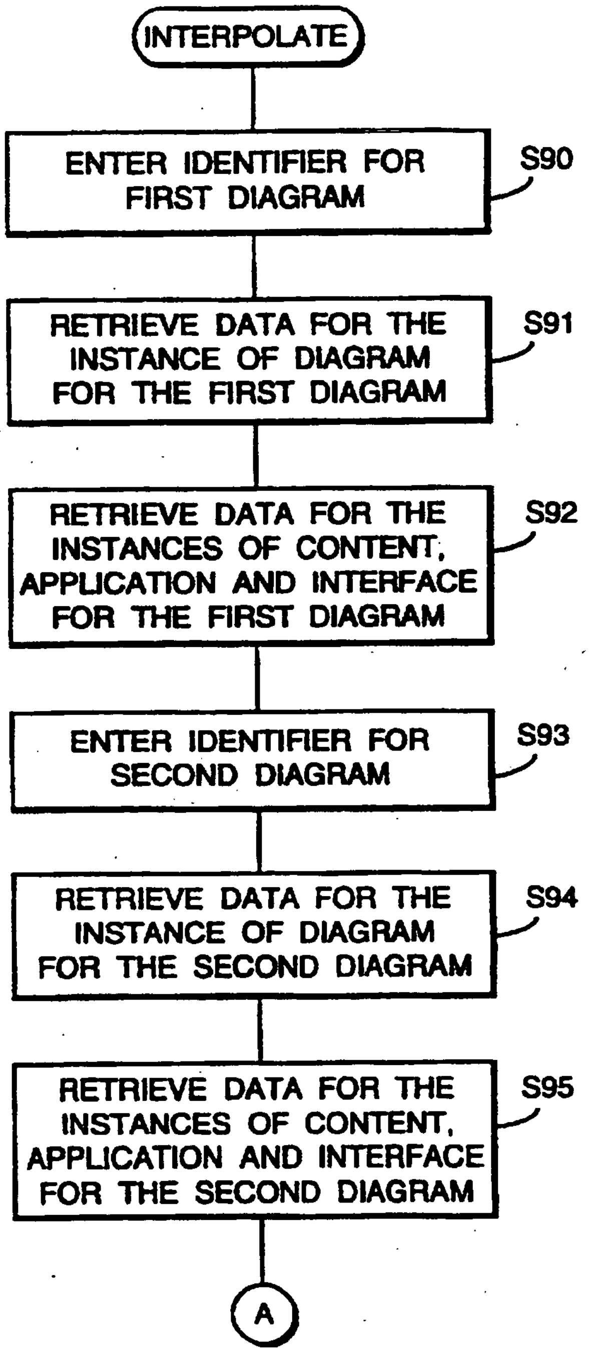

INTERPOLATE
ENTER IDENTIFIER FOR FIRST DIAGRAM
S90
RETRIEVE DATA FOR THE INSTANCE OF DIAGRAM FOR THE FIRST DIAGRAM
S91
RETRIEVE DATA FOR THE INSTANCES OF CONTENT, APPLICATION AND INTERFACE FOR THE FIRST DIAGRAM
S92
ENTER IDENTIFIER FOR SECOND DIAGRAM
S93
RETRIEVE DATA FOR THE INSTANCE OF DIAGRAM FOR THE SECOND DIAGRAM
S94
RETRIEVE DATA FOR THE INSTANCES OF CONTENT, APPLICATION AND INTERFACE FOR THE SECOND DIAGRAM
S95
A

Fig.13b.

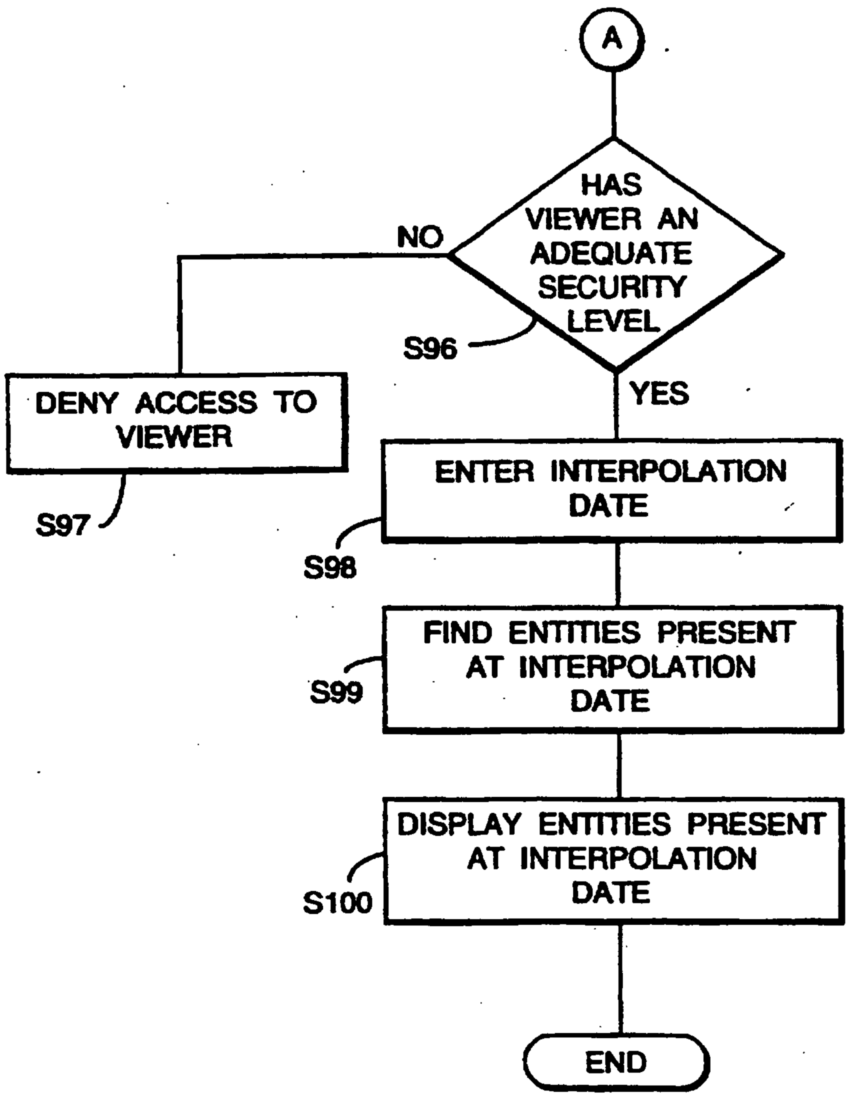

A
HAS VIEWER AN ADEQUATE SECURITY LEVEL
NO
YES
S96
DENY ACCESS TO VIEWER
S97
ENTER INTERPOLATION DATE
S98
FIND ENTITIES PRESENT AT INTERPOLATION DATE
S99
DISPLAY ENTITIES PRESENT AT INTERPOLATION DATE
S100
END

Fig.14.

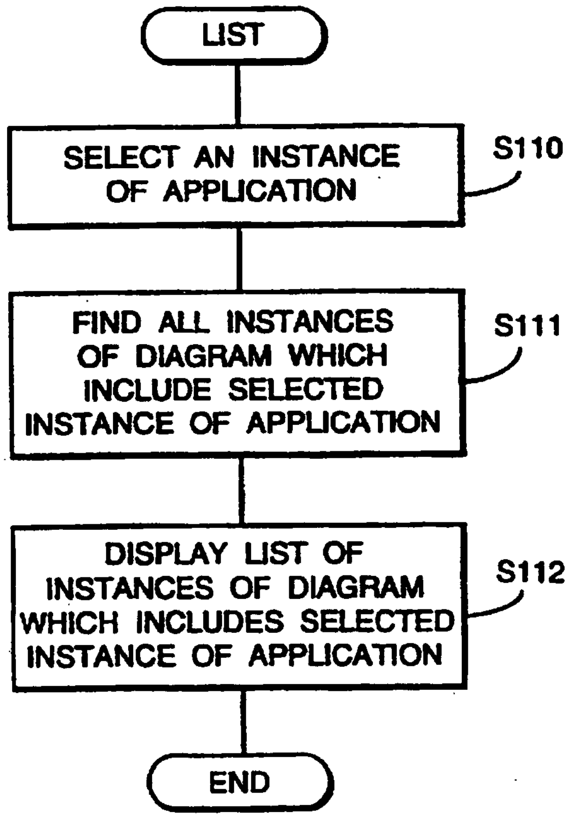

LIST
SELECT AN INSTANCE OF APPLICATION
S110
FIND ALL INSTANCES OF DIAGRAM WHICH INCLUDE SELECTED INSTANCE OF APPLICATION
S111
DISPLAY LIST OF INSTANCES OF DIAGRAM WHICH INCLUDES SELECTED INSTANCE OF APPLICATION
S112
END

Fig.15.

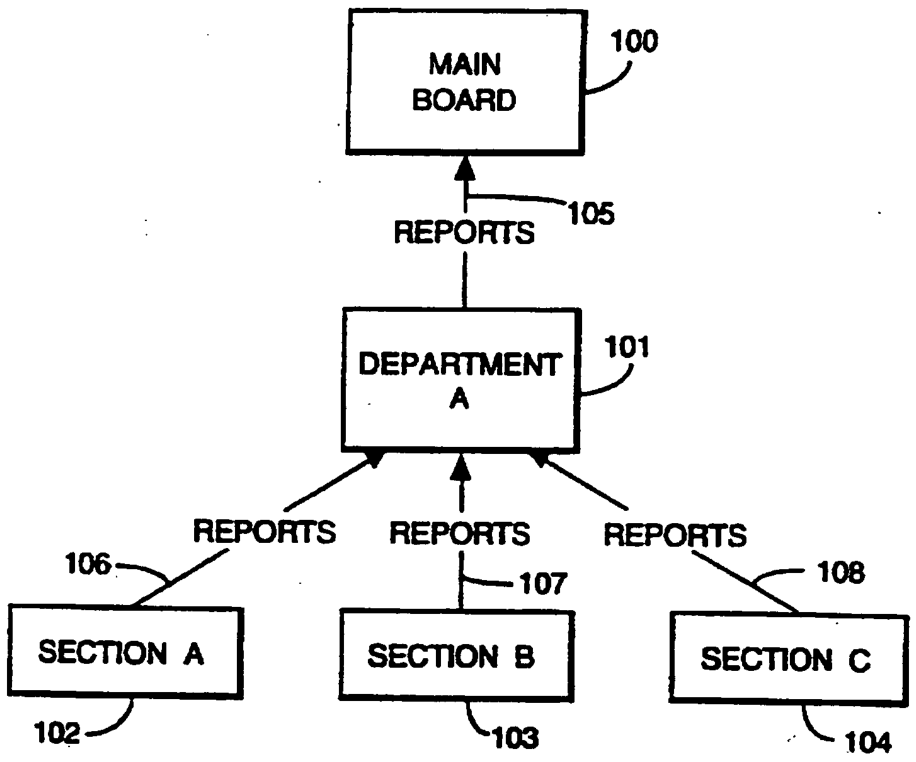

MAIN BOARD
100
105
REPORTS
DEPARTMENT A
101
REPORTS
REPORTS
REPORTS
106
107
108
SECTION A
SECTION B
SECTION C
102
103
104

Fig.16.

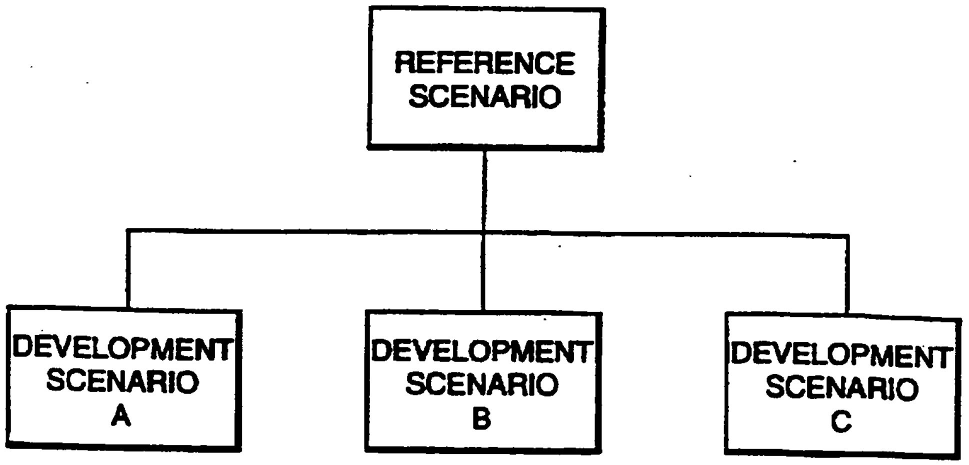

REFERENCE SCENARIO
DEVELOPMENT SCENARIO A
DEVELOPMENT SCENARIO B
DEVELOPMENT SCENARIO C

Fig. 17.

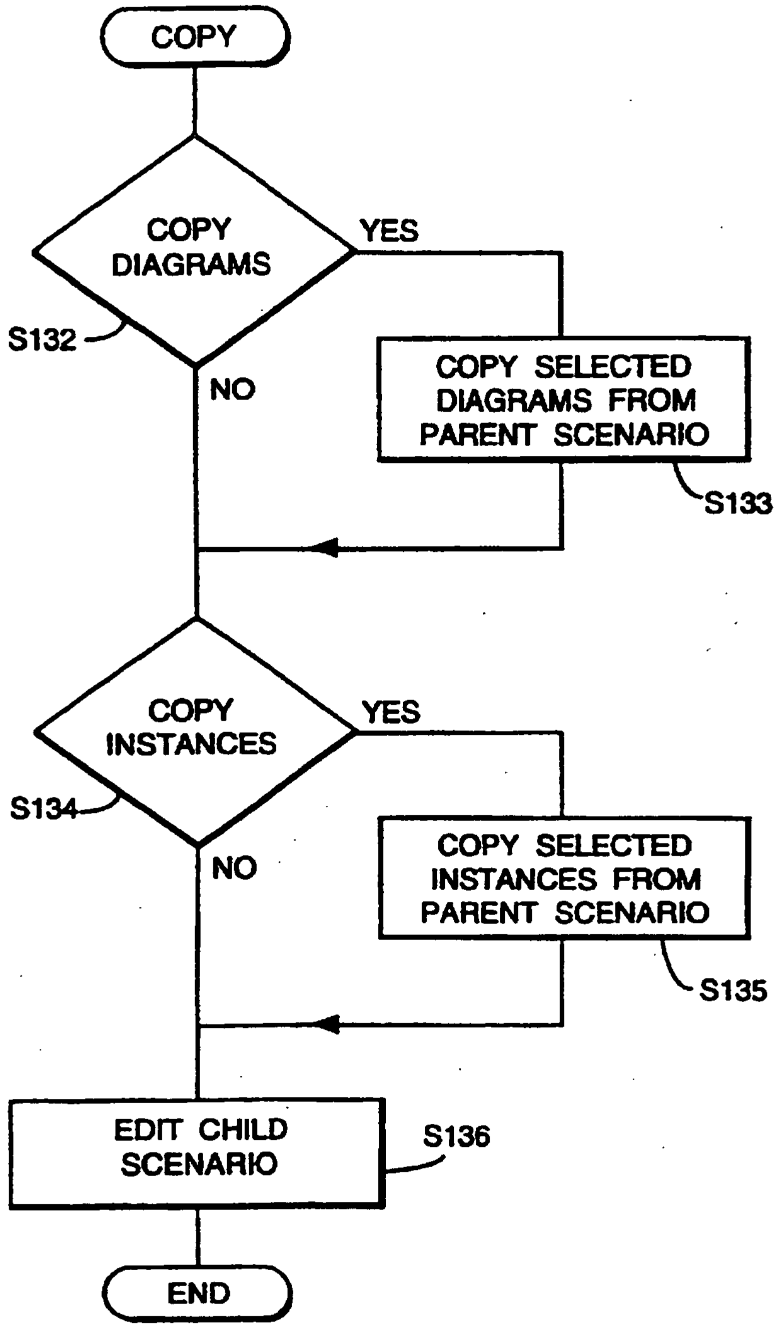

COPY
COPY DIAGRAMS
YES
NO
S132
COPY SELECTED DIAGRAMS FROM PARENT SCENARIO
S133
COPY INSTANCES
YES
NO
S134
COPY SELECTED INSTANCES FROM PARENT SCENARIO
S135
EDIT CHILD SCENARIO
S136
END

Fig.18.

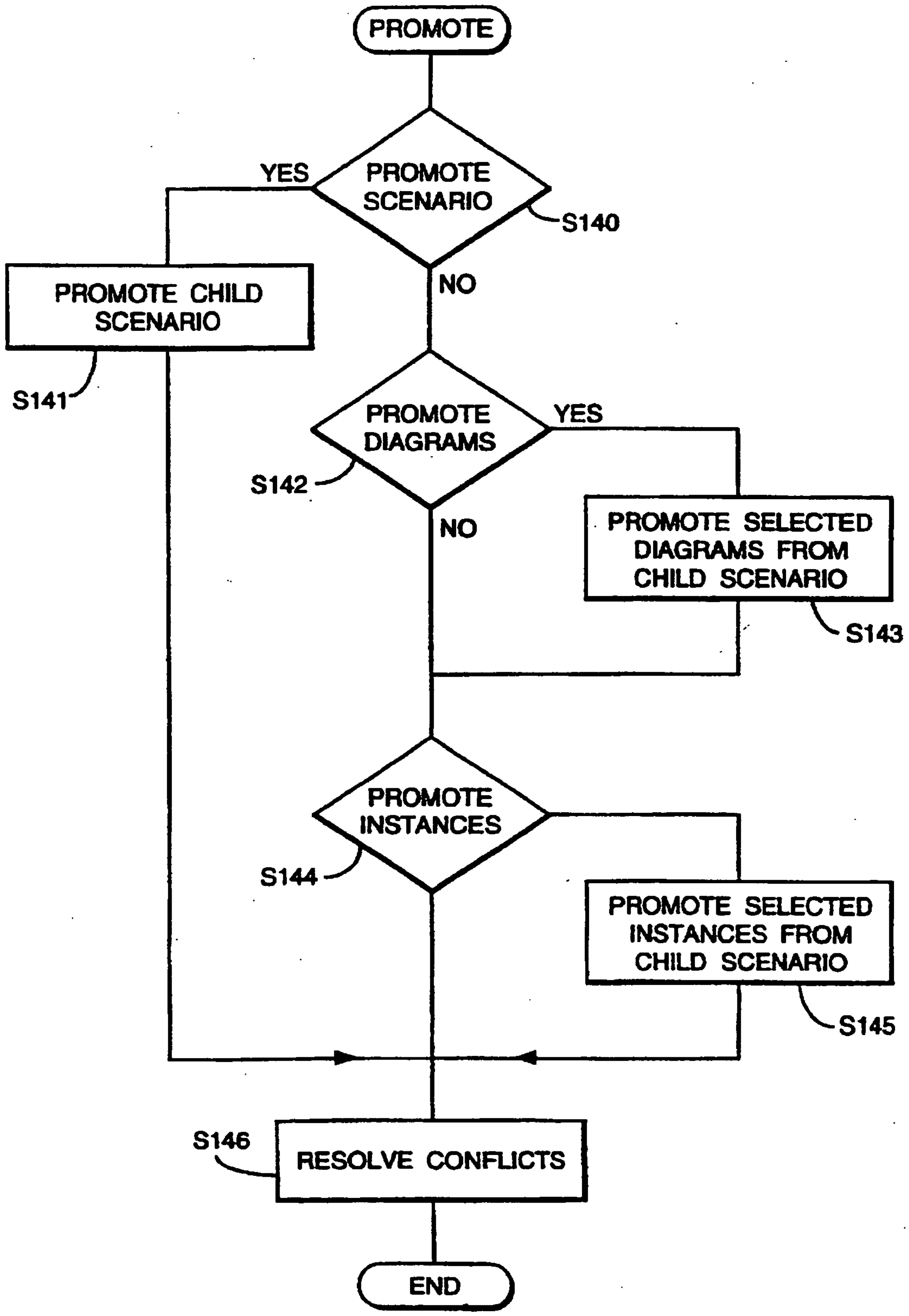

PROMOTE
PROMOTE SCENARIO
YES
S140
NO
PROMOTE CHILD SCENARIO
S141
PROMOTE DIAGRAMS
YES
S142
NO
PROMOTE SELECTED DIAGRAMS FROM CHILD SCENARIO
S143
PROMOTE INSTANCES
S144
PROMOTE SELECTED INSTANCES FROM CHILD SCENARIO
S145
S146
RESOLVE CONFLICTS
END

Fig.19.

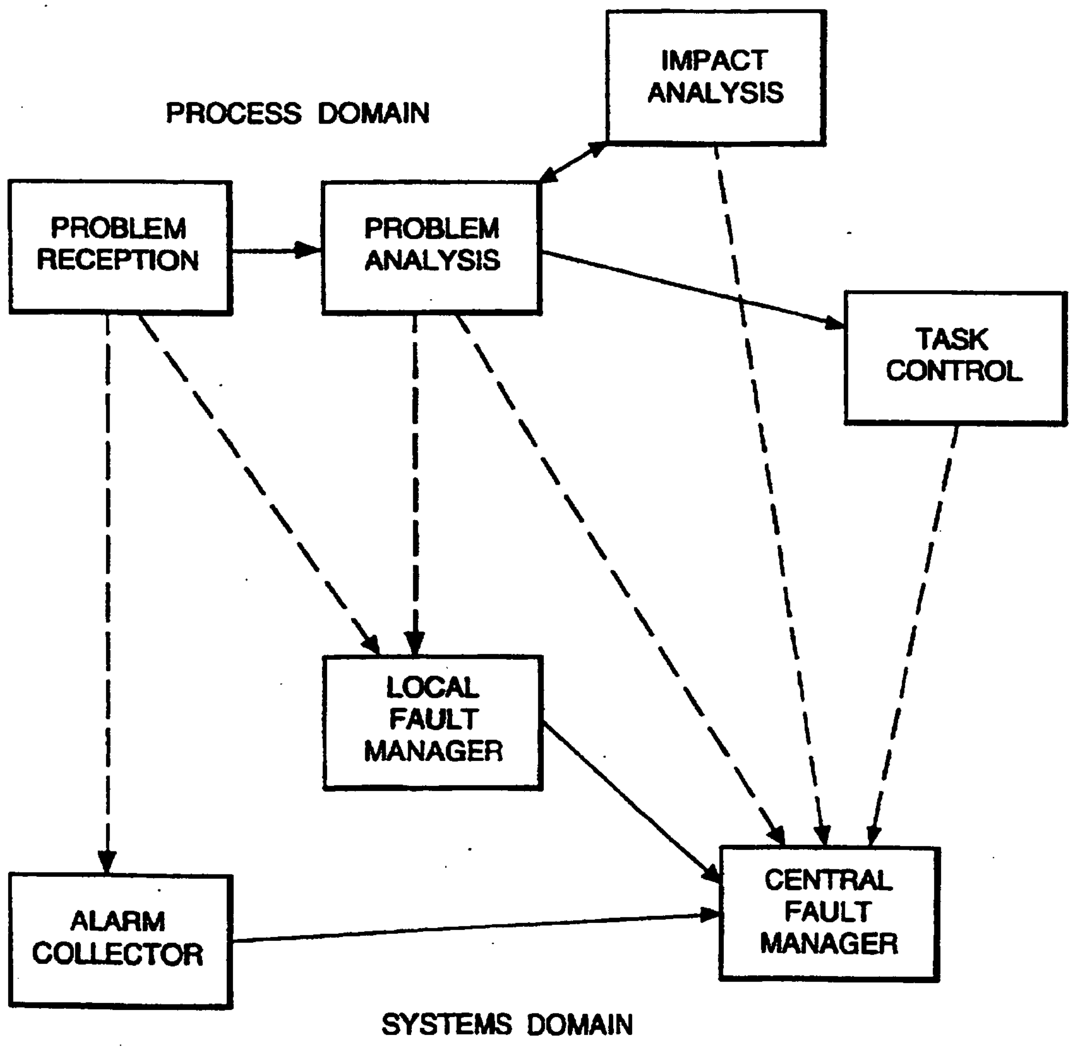

IMPACT ANALYSIS
PROCESS DOMAIN
PROBLEM RECEPTION
PROBLEM ANALYSIS
TASK CONTROL
LOCAL FAULT MANAGER
ALARM COLLECTOR
CENTRAL FAULT MANAGER
SYSTEMS DOMAIN

Fig.20a.

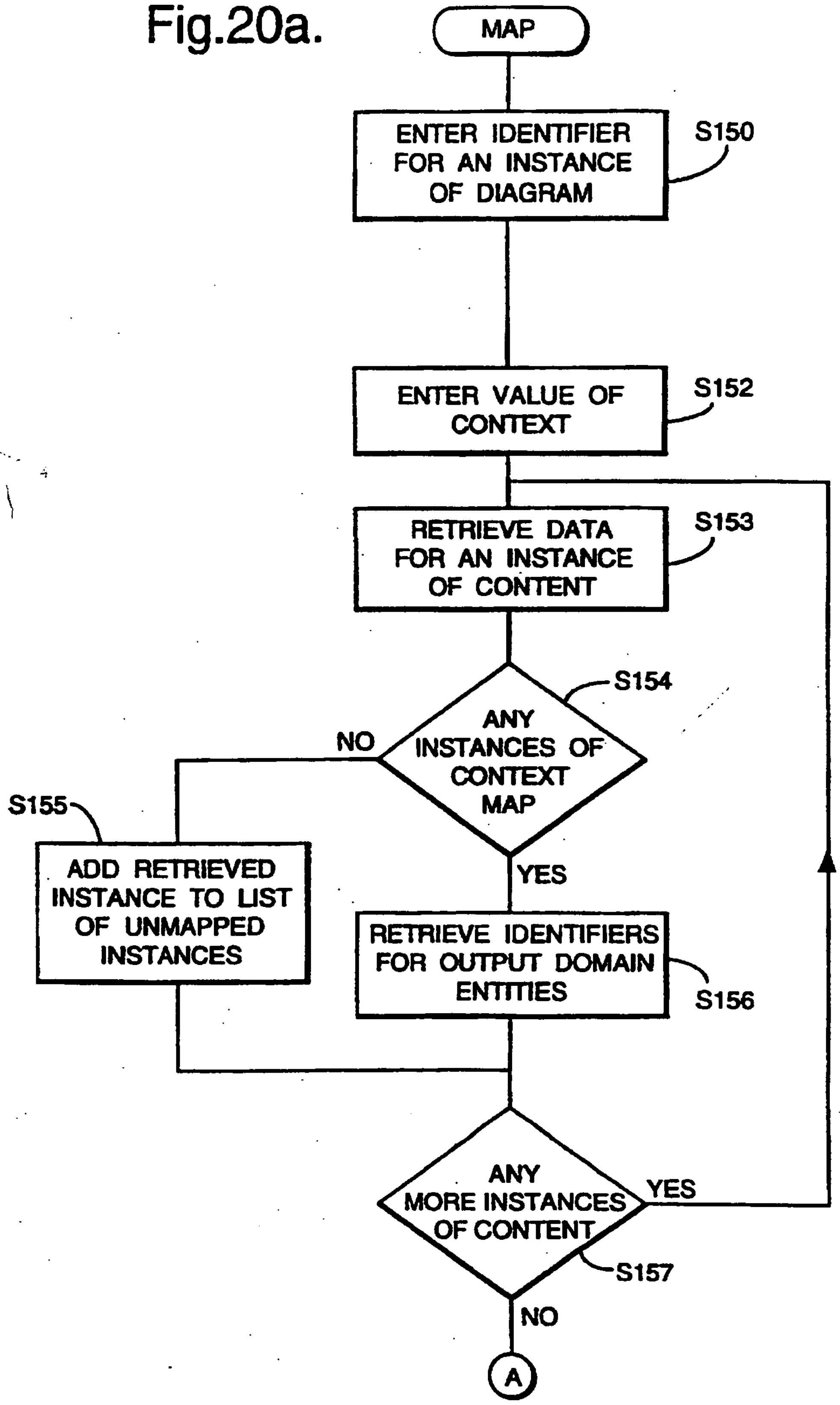
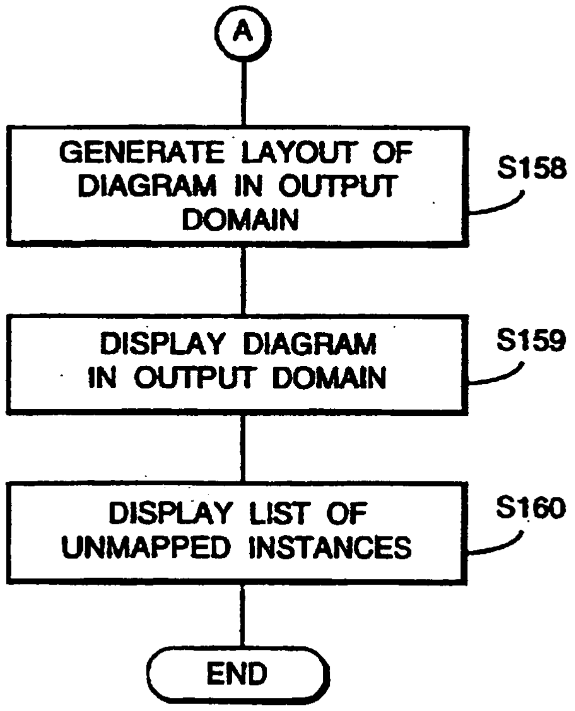

MAP
ENTER IDENTIFIER FOR AN INSTANCE OF DIAGRAM
S150
ENTER VALUE OF CONTEXT
S152
RETRIEVE DATA FOR AN INSTANCE OF CONTENT
S153
ANY INSTANCES OF CONTEXT MAP
S154
NO
S155
ADD RETRIEVED INSTANCE TO LIST OF UNMAPPED INSTANCES
YES
RETRIEVE IDENTIFIERS FOR OUTPUT DOMAIN ENTITIES
S156
ANY MORE INSTANCES OF CONTENT
YES
S157
NO
A

Fig.20b.


A
GENERATE LAYOUT OF DIAGRAM IN OUTPUT DOMAIN
S158
DISPLAY DIAGRAM IN OUTPUT DOMAIN
S159
DISPLAY LIST OF UNMAPPED INSTANCES
S160
END

APPARTUS AND METHOD FOR STORING DIAGRAM DATA

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of our copending commonly assigned PCT application No. PCT/GB95/00631 and it is also a continuation-in-part of our commonly assigned application Ser. No. 08/263,221 filed Jun. 21, 1994 now ABN.

This invention relates to an apparatus for storing data for use in displaying diagrams and also to a method of storing diagram data and using the stored data for displaying diagrams.

When storing data for use in displaying diagrams, it is desirable that the data can both be stored and retrieved in an efficient manner. In particular, it is desirable to be able to store data representing a particular diagram entity so that the diagram entity can be displayed in more than one diagram.

In EP-A-0 415 796 there is described an object-oriented display system which allows the creation of graphic diagrams. In this system, each diagram is created independently and there is no disclosure of using data representing a particular diagram entity in more than one diagram.

According to one aspect of this invention there is provided a computer apparatus for storing data for use in displaying diagrams, each diagram comprising a plurality of entities, said apparatus comprising:

means for entering data for use in displaying diagrams;

means for storing data describing an individual diagram entity as a software object having a set of attributes which includes an identifier for the software object;

means for storing general data relating to an individual diagram as a software object whose attributes include an identifier for the software object;

means for storing data relating to the location of an individual entity on a diagram as a software object whose attributes include the location of the entity on the diagram and a pointer to the software-object which describes the entity;

means for retrieving data describing the individual entities of a diagram; and

means for using the retrieved data for displaying a diagram.

The present invention provides the advantage that data describing a particular diagram entity can be used in displaying that diagram entity in more than one diagram.

The entities may include graphical shapes and flow lines connecting the graphical shapes together.

When the entities include graphical shapes and flow lines connecting the graphical shapes together, data describing an entity in the form of a graphical shape may be stored as a software object belonging to a software object class for graphical shapes, and data describing an entity in the form of a flow line may be stored as a software object belonging to a software object class for flow lines whose attributes include a pair of pointers which point to the identifiers of the two software objects which describe, respectively, the entity at one end and the entity at the other end of the flow line.

The apparatus may includes means for operating on the retrieved data in a predetermined manner to produce a desired diagram.

According to a second aspect of this invention, there is provided a computer apparatus including a central process-

ing unit, a memory, means for entering data and a display unit, said memory containing a program for controlling the computer apparatus and which is arranged to:

receive data for use in displaying diagrams;

store data representing an individual diagram entity as a software object having a set of attributes which includes an identifier for the software object;

store general data relating to an individual diagram as a software object whose attributes include an identifier for the software object;

store data relating to the location of an individual entity on a diagram as a software object whose attributes include the location of the entity on the diagram and a pointer to the software object which describes the entity;

retrieve data describing the individual entities of a diagram; and

use the retrieved data for displaying a diagram.

According to a third aspect of this invention, there is provided a method of operating a computer apparatus for storing diagram data and using the stored data for displaying diagrams, each diagram comprising a plurality of entities, said method comprising the steps of:

storing data describing each individual diagram entity as a software object having a set of attributes which includes an identifier for the object;

storing general data relating to each individual diagram as a software object whose attributes include an identifier for the software object;

storing data relating to the location of each individual entity of a diagram as a software object whose attributes include the location of the entity on the diagram and a pointer to the software object which describes the entity;

retrieving data describing the individual entities of a diagram; and

using the retrieved data for displaying a diagram.

This invention will now be described in more detail, by way of example, with reference to the accompanying drawings in which:

FIG. 1 is a block diagram of an apparatus embodying this invention;

FIG. 2 is a block diagram showing the components of a computer forming part of the apparatus;

FIG. 3 shows the components of the software used in the apparatus of FIG. 1;

FIG. 4 is a diagram illustrating some of the components of an alarm management system for a telecommunications network which are present on a particular day;

FIG. 5 is a flow chart of a routine STORE forming part of the main program of the apparatus of FIG. 1;

FIG. 6 including FIGS. 6*a* and 6*b* is a flow chart of a routine RETRIEVE used in the main program;

FIG. 7 is a diagram showing the network management centre together with four alarm collection centres which belong to the alarm management system of FIG. 4;

FIG. 8 including FIGS. 8*a* and 8*b* is a flow chart of a routine LINK forming part of the main program;

FIG. 9 is a diagram of the alarm management system of FIG. 4 and generally similar in layout to the diagram of FIG. 4 but showing the components which are present at a later date;

FIG. 10 is a diagram formed by combining FIGS. 4 and 9;

FIG. 11 including FIGS. 11*a* and 11*b* is a flow chart of a routine ADD forming part of the main program;

FIG. 12 is a flow chart of the alarm management system of FIG. 4 and generally similar to FIG. 4 but showing the components which are present at a date between the dates of the diagrams of FIGS. 4 and 9;

FIG. 13 including FIGS. 13*a* and 13*b* is a flow chart of a routine INTERPOLATE forming part of the main program;

FIG. 14 is a flow chart of a routine LIST forming part of the main program;

FIG. 15 is a diagram illustrating part of the management structure of a company;

FIG. 16 illustrates how stored data may be partitioned into scenarios;

FIG. 17 is a flow chart of a routine COPY which may form part of the main program;

FIG. 18 is a flow chart of a routine PROMOTE which may form part of the main program;

FIG. 19 illustrates the mapping between two diagrams representing a real world structure in two different domains; and

FIG. 20 including FIGS. 20*a* and 20*b* is a flow chart of a routine MAP which may form part of the main program.

Referring now to FIG. 1, there is shown an apparatus 10 for storing data for use in displaying diagrams. The apparatus 10 comprises three general purpose computers or workstations 11, 12, 13 connected by a communication link 14 to a file server 15. The file server 15 functions as a centralised data store for all three computers 11, 12, 13 and data for storage may be entered either at one of the computers 11, 12, 13 or directly into the file server. By providing three computers 11, 12, 13, three people may use the apparatus simultaneously and access the data stored in the file server 15. However, where it is required that only one person should be able to use the apparatus at any one time, it is sufficient to provide only one computer and the apparatus will be described below with reference to only computer 11.

The computer 11 is of conventional construction and its main components are shown in FIG. 2. These components include a mouse 19, a keyboard 20, a visual display unit or screen 21, a central processing unit (CPU) 22, a read only memory (ROM) 23, and a random access memory (RAM) 24. In operation, programs and data may be loaded from the file server 15 into RAM 24.

Referring now to FIG. 3, there are shown the components of the software or programs used in the apparatus 10. These components comprise a graphical user interface (GUI) 30, a main program 31, an interface 32 and a database management system 33. GUI 30 is formed from the two software packages AIDA and MASAI available from Ilog Limited of the Surrey Technology Centre, 40 Occam Road, Guildford, GU2 5YH, England. As the construction of GUIs is well known to those skilled in the art, GUI 30 will not be described in further detail.

The database management system 30 is the well known Oracle Database Management System. The main program 31 comprises routines STORE, RETRIEVE, LINK, ADD, INTERPOLATE and LIST which are described in detail below. In the present example, the main program 31 is written in the programming language LISP. The interface 32 functions as an interface between the main program 31 written in LISP and the Oracle Database management system 33. The interface 32 is the software package Asquell available from Ilog Limited.

The software components shown in FIG. 3 are stored in the file server 15 and may be loaded into the computer 11 when it is desired to use the apparatus.

Referring now to FIG. 4, there is shown a diagram illustrating some of the components of an alarm management system for a telecommunications network which are present at a certain date. In the present example, these components comprise switches A, B, C, alarm collection centre A, repair team A and a network management centre. In FIG. 4 these components are represented, respectively, by boxes 40 to 45. Each of these boxes is labelled with text which provides information, such as the name, about the component which it represents. As represented by flow lines 50, 51, 52, each of the switches A, B, C sends alarms to the alarm collection centre A. As illustrated by flow lines 53, 54, the alarm collection centre A sends reports to repair team A and the network management centre. Each of the flow lines 50 to 54 has a source and a destination. The source is located at the box which represents the component where information in the form of reports or alarms originates and the destination is connected to the box which represents the component to which the information is delivered. Each of the flow lines has an arrow at its destination. As may be observed, each of the flow lines is provided with text ("reports" or "alarms") indicating the name of the information which it carries.

The boxes 40 to 45 belong to one type of diagram entity while the flow lines 50 to 54 belong to another type of diagram entity.

The apparatus 10 operates in what is known as an object-oriented environment. In this environment, abstract or physical real world objects are modelled by software objects. The real world objects may be divided into object types. For example, the diagram boxes 40 to 45 belong to one type of real world object and the diagram flow lines 50 to 54 belonging to another type of real world object. The software implementation of an object type is known as an object class. A particular example of a software object class is known as instance of the object class or, more simply, as an object. Each software object class has a set of attributes. In the case of an instance of an object class, the attributes have values which collectively describe features of interest of the real world object which it models. The apparatus 10 uses four software object classes and these are: APPLICATION, INTERFACE, DIAGRAM and CONTENT. The attributes for these software classes are listed in Tables 1 to 4 below and these tables will now be described in turn. The attributes for the object class APPLICATION are shown in Table 1 below:

TABLE 1

Name of object class: APPLICATION

Attribute	Description
ID	unique identifier for entity
NAME	name of entity
START-DATE	start date of entity
END-DATE	end date of entity
VIEWER	security category for viewer

The object class APPLICATION is used to describe a diagram entity in the form of a box. For a particular instance of the object class APPLICATION, the attributes are set as follows. ID is a unique identifier for the entity. NAME is both the name of the real world object represented by the box and also the text appearing inside the box. START-DATE and END-DATE together define the period of validity or existence of the real world object represented by the box. VIEWER gives the security level required by a viewer or user to see or edit a diagram which includes the box.

In the present example, the object class APPLICATION is the only object class for describing a graphical shape. More generally, there may be other object classes for other graphical shapes such as circles and ellipses. Each object class may also be associated with a certain type of data, for example data relating to a monitoring system, a group of people or a database. If desired, the graphical shape assigned to a particular class may be changed by the user.

The attributes for the object class INTERFACE are shown below in Table 2.

TABLE 2

Name of object class: INTERFACE

Attribute	Description
ID	unique identifier for entity
NAME	name of entity
VIEWER	security category for viewer
SOURCE-CLASS-ID	class of entity at source of line
SOURCE-ENTITY-ID	identifier of entity at source of line
DEST-CLASS-ID	class of entity at destination of line
DEST-ENTITY-ID	identifier of entity at destination of line

An instance of INTERFACE describes a diagram entity in the form of a flow line. For each instance of this object class, the attributes are set as follows. ID and VIEWER are set in a similar manner to the corresponding attribute for an instance of APPLICATION. NAME is set to the text which appears on the flow line. SOURCE-CLASS-ID and SOURCE-ENTITY-ID are set to the name of the class and the identifier for the instance of that class for the diagram entity which appears at the source of the flow line. In the present example, the class is always APPLICATION but, more generally, flow lines could be connected to diagram entities of other classes for other graphical shapes such as circles or ellipses. DEST-CLASS-ID and DEST-ENTITY-ID give the name of the class and the identifier for the instance of that class for the diagram entity at the destination of the flow line.

The attributes for the class DIAGRAM are given in Table 3 below.

TABLE 3

Name of object class: DIAGRAM

Attribute	Description
ID	unique identifier for diagram
NAME	name of diagram
VIEW-DATE	date of diagram
VIEWER	security category for viewer

Each instance of DIAGRAM provides general data relating to a particular diagram. A description of the attributes is set out clearly in Table 3.

The attributes for the object class CONTENT are set out below in Table 4.

TABLE 4

Name of object class: CONTENT

Attribute	Description
ID	unique identifier for this entity on the diagram
X-COORD	x co-ordinate of this entity on the diagram
Y-COORD	y co-ordinate of this entity on the diagram

TABLE 4-continued

Name of object class: CONTENT

Attribute	Description
BOX-TYPE	type of box
LINE-TYPE	type of line
VIEWER	security category for viewer
DIAGRAM-ID	pointer to identifier for diagram
CLASS-ID	pointer to object class of entity
ENTITY-ID	pointer to unique identifier of entity

Each instance of CONTENT gives the location and other details of an entity of a diagram on that diagram. Thus, X-COORD and Y-COORD give the x and y co-ordinates of the entity on the diagram. In the present example, the entity can be only a box or a flow line. If it is a box, BOX-TYPE describes the type of box. For example, the box may be rectangular with straight sides or rectangular with curved sides. LINE-TYPE describes the type of line, for example thick or thin, used to draw the box or the flow line. DIAGRAM-ID gives the unique identifier for an instance of DIAGRAM which contains general data relating to the diagram on which this entity is present. CLASS-ID and ENTITY-ID together define the instance of the object class which describes this entity. Thus, if the entity is a box, CLASS-ID is set to APPLICATION. If it is a flow line, then CLASS-ID is set to INTERFACE.

The routine STORE is used for storing data for subsequent use in displaying a diagram. The flow chart for this routine is shown in FIG. 5 and the flow chart will be described with reference to storing the diagram of FIG. 4.

In a step S1, general data for the diagram is entered and stored as an instance of DIAGRAM.

Then, in a step S2, data describing one of the boxes 40 to 45, for example box 40, is entered and stored as an instance of APPLICATION. In a step S3, data describing the location of the box and other associated data is entered and stored as an instance of CONTENT. Steps S2 and S3 are then repeated until an instance of APPLICATION and an instance of CONTENT have been created for each of the boxes 40 to 45.

In a step S4, data is entered which describes one of the flow lines, for example flow line 50, and stored as an instance of INTERFACE. Then in a step S5, data describing the location of the flow line and other associated data is stored as an instance of CONTENT. Steps S4 and S5 are repeated until an instance of INTERFACE and an instance of CONTENT has been created for each of the flow lines 50 to 54.

Where data is being stored for a whole set of diagrams, it is possible that some of the boxes and flow lines may appear in more than one diagram. Where this is the case, for each box or flow line which is present in more than one diagram, it is necessary to create only one instance of APPLICATION or INTERFACE. If the data for a particular instance of APPLICATION or INTERFACE is subsequently updated, this is effective for all diagrams which include this instance.

The routine RETRIEVE is used for retrieving data for displaying a diagram. The flow chart for this routine is shown in FIG. 6 and this will now be described with reference to the diagram of FIG. 4.

In a step S20, the identifier for the instance of DIAGRAM relating to the diagram of FIG. 4 is entered. Then, in a step S21, the instance of DIAGRAM is retrieved.

In a step S22, an instance of CONTENT is retrieved which points to the instance of DIAGRAM retrieved in step

S21. The instance of CONTENT retrieved in S22 points to an instance of APPLICATION or INTERFACE. In S23, this instance of APPLICATION or INTERFACE is retrieved. In a step S24, a check is made to determine if there are any more instances of CONTENT. Steps S22 and S23 are repeated for each remaining instance of CONTENT which points to the instance of DIAGRAM retrieved in step S21 and for the associated instance of APPLICATION or INTERFACE.

At this point in executing the routine RETRIEVE, all the instances of CONTENT, APPLICATION and INTERFACE for the boxes 40 to 45 and flow lines 50 to 54 will be retrieved.

Each user of the apparatus has an associated security level. The security level will normally be allocated to each user by the person responsible for managing the apparatus. In a step S25, a check is made to determine if the user has an adequate security level to view the diagram. This is achieved by comparing the user's security level with the value of VIEWER for each instance of APPLICATION, INTERFACE, DIAGRAM and CONTENT which has been retrieved. If any one of these instances requires a security level greater than that possessed by the user, the user is denied access to the diagram. This is achieved by putting a suitable caption onto the display screen in a step S26 and then terminating execution of the routine.

If the user has an adequate security level, in a step S27, the diagram is displayed using the data which has been retrieved.

In a step S28, the user is asked if he wishes to edit the displayed diagram. If the user indicates that he does not wish to edit the displayed diagram, the routine is terminated. If the user indicates in step S28 that he wishes to edit the displayed diagram, the diagram is edited in step S29 and the routine is then terminated. The procedure of step S29 for editing the diagram will now be described.

In step S29, the user may edit the instance of DIAGRAM retrieved in step 21, any of the instances of CONTENT retrieved in step S22 and any of the instances of APPLICATION or INTERFACE retrieved in step S23. The user may edit any of these instances by displaying the attributes and their values and then changing the attribute value by using the keyboard 20. Alternatively, and preferably, the user may edit an instance by positioning a pointer over the relevant diagram entity with the aid of mouse 19. The user then clicks a button on the mouse and this causes an edit menu to be displayed. The user then edits the relevant instance by following the instructions on the edit menu.

When editing the instance of DIAGRAM retrieved in step S21, the user might change the security category of the viewer or the name of the diagram.

When editing one of the instances of CONTENT retrieved in step S22, the user might change its location by altering the values for its x and y co-ordinates. The user could also change the entity (box or flow-line) which is displayed by changing the pointer for the diagram entity. The user could also transfer the displayed entity to another diagram by altering the pointer for the diagram identifier.

When editing one of the instances of APPLICATION or INTERFACE retrieved in step S23, the user might change its name. In the case of an instance of INTERFACE, the user could change the identifier for the entity at its source or the identifier for the entity at its destination.

If an instance of APPLICATION or INTERFACE is edited in step S29, the changes will be effective in all diagrams in which the instance is used.

In step S29, a user may also delete or add instances of CONTENT, APPLICATION, INTERFACE and DIAGRAM.

Each of the classes APPLICATION, INTERFACE, DIAGRAM and CONTENT includes the attribute VIEWER which is used to specify the security of the viewer or user. In the present example, the same security category is used for both viewing and editing. Thus, if a user has an adequate security level to view a diagram, he may also edit the diagram. By way of modification, there may be separate security categories for viewing and editing. This would be achieved by providing two attributes in each class for security categories, one for viewing and one for editing. Where there are separate security categories for viewing or editing, a user may be given authority to view a diagram without being given authority to edit it.

When the present invention is used in a windowing environment, two or more diagrams may be displayed simultaneously on screen 21. Also specific diagram entities may be dragged by using mouse 19 from one diagram to another. When a diagram entity is dragged from one screen to another, it may be deleted from the first diagram and added to the second diagram. Alternatively, the diagram entity may be copied so that it remains in the first diagram and is added to the second diagram. This is achieved by creating and deleting instances of CONTENT as appropriate.

When a diagram entity is dragged from one diagram to another, the manner in which it is displayed may change. Thus, the shape, colour and scale of a diagram entity may change when it is dragged from one diagram to another. This is achieved by providing a set of display attributes for each class of diagram entity and providing values for these attributes for each instance of DIAGRAM. Thus, the display attributes could include an attribute COLOUR for specifying the colour in which an entity is displayed. Thus, for a particular instance of APPLICATION, the attribute COLOUR could be set to a value red for one instance of DIAGRAM and to a value green for another instance of DIAGRAM.

The ability to display two or more diagrams simultaneously and to drag diagram entities from one diagram to another may be achieved by using the following software: the Solaris operating system from SUN Microsystems Ltd, 31–41 Pembroke Broadway, Camberley, Surrey, GU15 3XD, together with the software package MASAI and LeLisp from Ilog Limited, Surrey Technology Centre, 40 Occam Road, Guildford, GU2 5YH.

When retrieving the data contained in the software objects, it is retrieved from the file server 15 and stored in RAM 24 of computer 11.

Referring back to FIG. 4, this figure shows the box 45 connected by flow line 54 to box 43. In other diagrams stored in the file server 15, there may be other boxes connected by other flow lines to the box 45. The routine LINK is capable of finding these other boxes and flow lines and generating a diagram which shows all the boxes and flow lines connected to box 45. An example of a diagram which may be produced by the routine LINK is shown in FIG. 7.

FIG. 7 shows box 43 connected by flow line 54 to box 45. This information is, of course, present on FIG. 4. FIG. 7 also shows boxes 60, 61, 62 connected by flow lines 63, 64, 65 to box 45. Boxes 60, 61 and 62 represent three further alarm collection centres, namely, alarm collection centres B, C and D. The flow chart for the routine LINK is shown in FIG. 8

and this flow chart will be described with reference to generating the diagram shown in FIG. 7.

The user initially selects the box which is of interest. In the present example, box 45 is selected. Then, in a step S40, the user enters the identifier for the instance of APPLICATION containing data describing box 45. The user may perform step S40 by typing in the characters of the identifier. Alternatively, there may already be a diagram displayed on the screen which shows box 45 and the user then positions a pointer over box 45 with the aid of the mouse 19 and clicks a button to indicate that this is the selected box.

In a step S41, the selected instance of APPLICATION is retrieved.

In a step S42, an instance of INTERFACE which describes a flow line connected to box 45 is retrieved. In the present example, this may be the instance of INTERFACE which describes the flow line 54. In a step S43, the instance of APPLICATION which describes the box connected to the other end of the flow line is retrieved. For example, if the instance of INTERFACE retrieved in step S42 describes flow line 54, then the instance of APPLICATION which describes box 43 is retrieved in step S43. In a step S44, a check is made to determine if there are any more instances of INTERFACE. Steps S42 and S43 are then repeated for all remaining instances of INTERFACE which describe flow lines connected to box 45 and the corresponding remaining instances of APPLICATION which describe the boxes connected to the other ends of these flow lines.

In step S45, a check is made to determine if the user has an adequate security level to view the boxes and flow lines which will be displayed. This step is similar to step S24 described with reference to FIG. 6. If the user does not have an adequate security level, access is denied to him in a step S46.

If the user has an adequate security level, in a step S47, the retrieved instances of APPLICATION are counted. In the present example, there are four such instances. In a step S48, the diagram is formatted. This is achieved by positioning the selected box at the centre of the diagram and then positioning the flow lines and boxes which are connected to the selected box at equal angular spacings around the selected box. In the present example, as there are four boxes connected to the selected box, they are displayed at 90° angles around the selected box.

In a step S49, the new diagram is displayed.

In the routine LINK, the instances of APPLICATION and INTERFACE retrieved in steps S41, S42 and S43 are retrieved from the file server 15 and loaded into RAM 24 of computer 11. Steps S45, S47, S48 and S49 are then performed with the data in RAM 24.

FIG. 9 shows the alarm management system of FIG. 4 at a later date. At the date shown in FIG. 9, switch A no longer exists and a new switch, namely switch D, has been added. In FIG. 9, switch D is represented by box 70 and flow line 71 shows alarms being passed from switch D to alarm collection centre A.

The apparatus 10 is capable of combining two diagrams representing a particular real world arrangement, such as the alarm management system of FIGS. 4 and 9, at two different dates. For example, it can combine the diagrams of FIGS. 4 and 9 to produce the diagram shown in FIG. 10. In FIG. 10, the boxes and flow lines which are present in both FIGS. 4 and 9 are shown in a first manner, namely by solid lines. The box 40 and flow line 50, which are present only in FIG. 4, are shown in a second manner, namely by dashed lines. The box 70 and the flow line 71, which are shown only in FIG. 9, are shown in the diagram of FIG. 10 in a third manner, namely with chain dotted lines. On a colour display screen, the solid, dashed and chain dotted lines of FIG. 10 may be replaced by three different colours.

The operation of combining two diagrams is performed by the routine ADD and the flow chart for this routine is shown in FIG. 11. The flow chart of FIG. 11 will now be described with reference to FIGS. 4, 9 and 10.

At the start of the routine, in a step S70 the user enters the identifier for the instance of DIAGRAM which contains general data for the first diagram. Thus, in the present example, the user enters the identifier for the instance of DIAGRAM for the diagram shown in FIG. 4. Then in a step S71, this instance of DIAGRAM is retrieved.

In a step S72, the instances of CONTENT, APPLICATION and INTERFACE for the first diagram are retrieved. Step S72 thus corresponds generally to steps S22 and S23 shown in FIG. 6.

Then, in a step S73, the user enters an identifier for the instance of DIAGRAM for the second diagram. In both steps S70 and S73, the identifier can be entered either by keying the characters which form the identifier or, if the diagrams are already displayed, by positioning a pointer over the diagram with the aid of mouse 19 and then clicking a button on mouse 19.

In steps S74 and S75, the instance of DIAGRAM and the instances of CONTENT, APPLICATION and INTERFACE for the second diagram are retrieved. In a step S76, a check is made to determine if the user has an adequate security level to view the boxes, flow lines and other data which will be shown in the eventual diagram. Thus step S76 corresponds generally to step S24 shown in FIG. 6. If the user does not have an adequate security level, access is denied to him in a step S77.

In a step S78, the diagram entities which are present in both the first and second diagrams are found. Step S78 is performed as follows. For each instance of CONTENT for the first diagram, it is determined if there is a corresponding instance of CONTENT for the second diagram which points to the same instance of APPLICATION or INTERFACE as the instance of CONTENT for the first diagram. The resulting instances of CONTENT, APPLICATION and INTERFACE are then associated with each other.

In a step S79, the diagram entities which are present in both diagrams are displayed in the first manner mentioned above. Thus, at this stage, boxes 41 to 45 and flow lines 51 to 54 are displayed as shown in FIG. 10.

In a step S80, the diagram entities which are present only in the first diagram are found. In order to achieve this, for each instance of CONTENT for the first diagram, it is determined if there is a corresponding instance of CONTENT for the second diagram which points to the same instance of APPLICATION or INTERFACE as the instance of CONTENT for the first diagram. If there is no corresponding instance of CONTENT for the second diagram, then the instance of CONTENT for the first diagram relates to a diagram entity present only in the first diagram. These instances of CONTENT together with the corresponding instances of APPLICATION and INTERFACE are then associated together.

Then, in a step S81, the diagram entities present only in the first diagram are displayed in the second manner. Thus, at this stage, box 40 and flow line 50 are added to the displayed diagram.

In a step S82, the diagram entities which are present only in the second diagram are found. This step is analogous to step S80.

Lastly, in a step S83, the entities present only in the second diagram are displayed in the third manner. Therefore, at this stage, box 70 and flow line 71 are added to the displayed diagram.

In steps S71, S72, S74 and S75, data is retrieved from the file server 15 and stored in RAM 24. Steps S76 to S83 are then performed on data in RAM 24. Because these steps are performed on data held in RAM 24, rather then in the file server 15, these steps are performed relatively quickly.

By way of modification, instead of combining FIGS. 4 and 9 to produce a new diagram in which the boxes and flow lines are represented in three different ways, the two diagrams may be kept separate. In this case, in both diagrams, the boxes and flow lines which are present in both diagrams are shown in a first manner, for example by solid lines. Boxes and flow lines which are present only in the first diagram are shown in the first diagram in a second manner, for example by dashed lines. Boxes and flow lines which are present only in the second diagram are shown in the second diagram in a third manner, for example by chain dotted lines.

Where file server 15 contains data for two diagrams having different dates but both showing the same real world arrangement, for example the alarm management system of FIG. 4, the apparatus 10 is capable of producing a further diagram showing those entities of the arrangement which are present at a specified date which is between the dates of the two diagrams. Thus, the apparatus is able to interpolate between two diagrams. FIG. 12 shows the result of interpolating between FIGS. 4 and 9 on a specified date. In the present example, FIG. 12 shows all the boxes and flow lines which are present in FIG. 4 and, in addition, shows box 70 and flow line 71 of FIG. 9. Thus, at the date of FIG. 12, switch D has been added but switch A has not yet been removed from the alarm management system.

The routine INTERPOLATE is responsible for interpolating between two diagrams and the flow chart for this routine in shown in FIG. 13. The flow chart of FIG. 13 will now be described with reference to FIGS. 4, 9 and 12.

After entering this routine, in a set of steps S90 to S95 corresponding to steps S70 to S75 shown in FIG. 11, the data for the two diagrams is retrieved from the file server 15 and loaded into RAM 24. Thus, in the present example, the data for both FIGS. 4 and 9 is retrieved in these steps. Subsequent steps are performed with the data in RAM 24.

In a step S96, corresponding to step S24 of FIG. 6, a check is made to determine if the user has an adequate security level to view the retrieved diagram entities. If the user does not have an adequate security level, access is denied in a step S97.

Then, in a step S98, the user enters the date of interpolation. In the present example, this will be the date of the diagram shown in FIG. 12.

Then, in a step S99, the diagram entities which are present at the date of interpolation are found. In order to achieve this, for each instance of CONTENT for the first diagram which points to an instance of APPLICATION, the values of START-DATE and END-DATE are compared with the date of interpolation. If the date of interpolation falls within the period covered by the values of START-DATE and END-DATE, then both the instance of CONTENT and the corresponding instance of APPLICATION are put on a list for display.

If the interpolation date does not fall within the period covered by the values of START-DATE and END-DATE, then the instance of CONTENT and the corresponding instance of APPLICATION are put on a list of entities which are not to be displayed.

Instances of CONTENT for the second diagram which point to instances of APPLICATION not present in the first diagram are then examined in a similar manner.

For each instance of APPLICATION which is on the list of entities which are not to be displayed, each instance of INTERFACE which describes a flow line connected to the box described by the instance of APPLICATION is found and also put on the list of entities which are not to be displayed.

In the present example, all of the diagram entities of FIG. 4 and in addition box 70 and flow line 71 of FIG. 9 are to be displayed and no diagram entities are excluded.

Then, in a step S100, the diagram entities present at the interpolation date are displayed.

Referring now to FIG. 14, there is shown the flow chart for the routine LIST. This routine is capable of producing a list of all the diagrams which contain a selected box.

After entering this routine, in a step 110, the user selects a box and enters the identifier for the instance of APPLICATION which describes the selected box. If a diagram is being displayed, the user may enter the identifier by clicking a button on mouse 19 with the mouse pointer over the selected box.

Next, in a step S111, the routine finds each instance of DIAGRAM which contains general data about a diagram which includes the selected box. This is achieved as follows. For each instance of DIAGRAM, all the instances of CONTENT which point to the instance of DIAGRAM are found. Each of these instances of CONTENT is examined to determine if it points to the instance of APPLICATION for the selected box. Then, in a step S112, a list is displayed of these instances of DIAGRAM.

If the user wishes to display a diagram corresponding to one of the instances of DIAGRAM on the list, the user may achieve this by using the routine RETRIEVE.

The diagram of FIG. 4 representing an alarm management system is only one example of the type of diagram which may be stored in the apparatus 10. FIG. 15 represents another type of diagram which may be stored and this figure shows part of the structure of a company together with reporting lines. Thus, FIG. 14 shows the main board represented box 100, department A represented by box 101, sections A, B, C represented by boxes 102, 103, 104, and the reporting lines represented by flow lines 105 to 108.

In a development of this invention, the diagram data is partitioned into scenarios and the scenarios are arranged in a hierarchical or parent-child structure. This development is suitable for modelling an evolving structure. By way of example, FIG. 16 shows how diagram data for modelling a telecommunications network may be partitioned into a parent scenario, namely a reference scenario, and three child scenarios, namely development scenarios A, B, C. The reference scenario contains diagram data for the network as presently existing and planned. The development scenarios contain diagram data for proposed developments to the network. If desired, the number of levels in the hierarchy may be increased, for example by creating child scenarios for development scenarios A, B and C.

With this development, a user of the system opts to work in a single scenario. Each user has a personal access profile which determines which scenarios he has permissions to read or edit. The scenario he selects defines the data which he can view or edit.

Each instance of APPLICATION, INTERFACE, DIAGRAM and CONTENT is owned by a single scenario. The

VIEWER attribute is modified to point to the owning scenario rather than an individual user. Instances owned by the parent scenario can be viewed by the child scenario, but instances owned by the child scenario cannot be viewed by the parent. In addition, instances of APPLICATION and INTERFACE which are owned by the parent scenario may be used as diagram entities in the child scenario. In this case, the DIAGRAM and CONTENT instances are owned by the child scenario, but ownership of the APPLICATION and INTERFACE instances remain with the parent scenario.

The attributes of an instance may only be edited whilst working in its owning scenario. However, an instance owned by the parent scenario may be modified in a child scenario by creating a copy of the instance which is owned by the child scenario. Instances created or modified in the child scenario may then be promoted to the parent scenario. In order to achieve this, routines COPY and PROMOTE are added to the main program 31. These two routines will now be described.

The routine COPY is used for copying diagram data from a parent scenario to a child scenario. Referring now to FIG. 17, after entering the routine COPY, in a step S132 a check is made to determine if the user wishes to copy the data for any of the diagrams owned by the parent scenario. If the user does wish to copy data for one or more of the the diagrams in the parent scenario, this is achieved in step S133.

In step S133, for each diagram selected by the user, data is copied for the respective instance of DIAGRAM and all instances of CONTENT which point to the copied instance of DIAGRAM. Each copied instance is given a new identifier so that a particular instance is owned by one scenario only. In addition, each copied instance is given a pointer which points to the corresponding instance in the parent scenario. The new instances of CONTENT point to instances of APPLICATION and INTERFACE which are owned by the parent scenario. After step S132 or step S133, the routine continues with step S134.

In step S134, a check is made to determine whether the user wishes to copy any individual instances of APPLICATION or INTERFACE from the parent scenario to the child scenario. If the user does wish to copy any of these instances, this is achieved in step S135. In step S135, the data for each selected instance is copied from the parent scenario to the child scenario. Each copied instance is given a new identifier so that a particular instance is owned by one scenario only. In addition, each copied instance is given a pointer which points to the corresponding instance in the parent scenario.

After step S134 or S135, the routine continues with step S136 in which the diagrams in the child scenario are edited. This is achieved in the manner described with reference to step S28 in FIG. 6. Although not shown, the use may return to steps S133 or S135 to copy further diagrams or instances if this is necessary during editing.

The routine PROMOTE is used for promoting diagram data from a child scenario to a parent scenario. This routine may be used to promote all the data owned by the child scenario to the parent scenario, or the data for selected diagrams to the parent scenario, or the data for selected instances to the parent scenario. This routine will now be described with reference to FIG. 18.

After entering the routine PROMOTE, in a step S140 a check is made to determine if the user wishes to promote all the data from the child scenario to the parent scenario. If the user does wish to transfer all the data, this is achieved in a step S141.

In step S141, the data for each instance in the child scenario is used to replace the existing data for the corresponding instance in the parent scenario. This is achieved by using the pointers in the instances in the child scenario which point to the corresponding instances in the parent scenario. Data for new instances in the child scenario is also added to the data contained in the parent scenario.

Deletion of instances in the child scenario is handled by marking the instances as DELETED, rather than physically removing them from the database. Consequently, if an instance is deleted in the child scenario, the corresponding instance in the parent scenario will be overwritten by an instance marked as DELETED. This 'soft deletion' method ensures that deletion of the instance is made visible to all scenarios which make use of the instance and gives users of these scenarios an opportunity to make any necessary changes. Periodically, the database is purged to permanently remove instances which have been marked as DELETED for a significant period, e.g. 6 months.

It is possible that there will be a conflict between data contained in the child scenario and data contained in the parent scenario. For example, promoting data from the child scenario to the parent scenario may result in a new diagram entity in the child scenario overlapping with an existing entity in the parent scenario. Any such conflicts are recorded in step S141 for resolution in step S146. The routine continues with step S146 after step S141.

If the user does not wish to promote the entire child scenario to the parent scenario, a check is made in step S142 to determine if the user wishes to promote the data for any of the diagrams of the child scenario to the parent scenario. If the user does wish to promote any of the individual diagrams, this is achieved in step S143.

In step S143, the data for the instance of DIAGRAM for the selected diagram, together with the data for the associated instances of APPLICATION, INTERFACE and CONTENT which are owned by the child scenario are used to replace the data for the corresponding instances in the parent scenario. Any conflicts are recorded for resolution in step S146. After step S142 or step S143, the routine continues with step S144.

In step S144, a check is made to determine if the user wishes to promote the data for any individual instances of APPLICATION or INTERFACE from the child scenario to the parent scenario. If the user does wish to promote the data for any of the individual instances, this is achieved in step S146.

In step S145, the user may select the instances to be promoted from an index of instances. Alternatively, a diagram from the child scenario may be displayed simultaneously with the corresponding diagram from the parent scenario. In this case, the individual instances are promoted by dragging the corresponding diagram entities with the aid of mouse 19 from the child diagram to the parent diagram.

After step S144 or step S145, the routine continues with step S146 in which conflicts are resolved. By way of example, in step S146, where a diagram entity in the child scenario has been found to overlap a diagram entity in the parent scenario, the conflict may be resolved by the user by deleting the diagram entity in the parent scenario.

In a further development, the present invention can generate a diagram representing a real world structure in one domain ("the output domain") from a diagram representing the same real world structure in another domain ("the input domain"). By way of example, this further development will be described with reference to a public telecommunications network in which the input domain is the process domain and the output domain is the systems domain. FIG. 19 shows, by way of example, a diagram in its upper half which represents fault management for part of the network in the process domain. In its lower half, FIG. 19 shows the corresponding diagram in the systems domain. The dashed lines show the links or mappings between the components of the diagram in the process domain and the components of the diagram in the systems domain.

In this further development, the object class APPLICATION is used to describe diagram entities in the form of boxes which represent components of the telecommunications network in the systems domain. With this further development, an additional object class PROCESS is used. The object class PROCESS is used to describe diagram entities in the form of boxes which represent components of the telecommunications network in the process domain. The attributes for the object class PROCESS are shown in Table 5 below:

TABLE 5

Name of object class: PROCESS

Attribute	Description
ID	Unique identifier for entity
NAME	Name of entity
VIEWER	Security category for viewer

Although in the present example there is only a single object class for describing diagram entities in each domain, in general there may be one or more object classes for describing diagram entities in each domain.

The links between diagram components in the input domain and diagram components in the output domain are held in instances of an object class CONTEXT MAP. More specifically, each instance of CONTEXT MAP holds a link between one diagram component in the input domain (for example, an instance of PROCESS) and one diagram component in the output domain (for example, an instance of APPLICATION). A diagram component in the input domain may have more than one link to diagram components in the output domain. For example, in FIG. 19, the diagram component named Problem Reception in the process domain has a link to Alarm Control and a link to Load Fault Manager in the output domain.

The attributes for the object class CONTEXT MAP are shown in FIG. 6 below.

TABLE 6

Name of object class: CONTEXT MAP

Attribute	Description
ID	Unique identifier for instance
INPUT_CLASS_ID	Pointer to object class of diagram entities in input domain
INPUT_INSTANCE_ID	Pointer to unique identifier for diagram entity in input domain
OUTPUT_CLASS_ID	Pointer to object class of diagram entities in output domain
OUTPUT_INSTANCE_ID	Pointer to unique identifier for diagram entity in output domain
CONTEXT	Context in which mapping occurs
VIEWER	Security category for viewer

The attributes ID and VIEWER are set in a similar manner to the corresponding attributes for the other classes described above. INPUT_CLASS_ID is set to the name of one of the object classes ("the input object class") used to describe the diagram entities which represent the components of the modelled structure in the input domain. In the present example, the input object class is PROCESS. INPUT_INSTANCE_ID is set to the value of the unique identifier for the instance ("the input instance") of this object class.

OUTPUT_CLASS_ID is set to the name of one of the object classes ("the output object class") used to describe the diagram entities which represent the components of the modelled structure in the output domain. In the present example, the output object class is APPLICATION. OUTPUT_INSTANCE_ID is set to the value of the unique identifier for the instance ("the output instance") of this object class.

CONTEXT specifies the context in which the mapping occurs. For example, for fault management in a public telecommunications network there may be one mapping for fault management in the trunk network and another mapping for fault management in the access network. Where mappings can occur only in a single context, the attribute CONTEXT is not needed. In the present example, it is assumed that the mapping is in the trunk network.

Referring now to FIG. 20, there is shown a flow chart for a routine MAP which is used to generate a diagram in an output domain from a diagram in an input domain.

After entering the routine MAP, in a step S150, the user enters the identifier for the instance of DIAGRAM which describes the diagram in the input diagram. In the present example, the user enters the identifier for the diagram shown in the upper half of FIG. 19. Then in a step S152, the user enters the value of the attribute CONTEXT. In the present example, the value indicates that the context for mapping from the input domain to the output domain is the trunk network.

Next, in a step S153, data is retrieved for an instance of CONTENT which points to the instance of DIAGRAM retrieved in step S150.

In step S154, it is determined if there are any instances of CONTEXT MAP for which the input instance is the instance found in step 153. If there are no such instances, the instance found in step S153 is put on a list of unmapped instances in a step S155.

If, in step S154, it is found that there are instances of CONTEXT MAP for which the input instance is the instance retrieved in step S153, the routine goes from step S154 to step S156. In step S156, for each instance of CONTEXT MAP for which the input instance is the instance retrieved in step S153, the identifier and the class name for the corresponding output instance are retrieved. Thus, in the present example, one or more identifiers for the object class APPLICATION are retrieved.

After step S155 or step S156, the routine continues with step S157. In this step, a check is made to determine if there are any more instances of CONTENT which point to the instance of DIAGRAM retrieved in step S150. If there are any such instances, the routine returns to step 153. If there are no such instances, the routine continues with step S158.

By the time the routine reaches step S158, in the present example there will have been retrieved a number of instances of APPLICATION. Some of these instances may have been retrieved several times. In the example shown in FIG. 19, providing all the relevant instances of CONTEXT MAP exist, the instance of APPLICATION for the box labelled Central Fault Manager will have been retrieved three times. In step S158, using an appropriate algorithm, these instances of APPLICATION are used to generate the layout for a diagram in the output domain. A suitable diagram is described in an article entitled "Graph Drawing by Force Directed Placement", Software Practice and Experience, Volume 21(11), pages 1129–1164, November 1991. This article is incorporated herein by reference.

Next, in a step S159, the output domain diagram generated in step S158 is displayed. However, if any unmapped instances have been found in step S154, the diagram will be incomplete.

Finally, in a step S160, the list of unmapped instances is displayed. If there are any unmapped instances, the user can then create appropriate instances of CONTEXT MAP so that the instances are no longer unmapped. The routine may then be run again so as to obtain a complete diagram in the output domain.

We claim:

1. A computer apparatus for storing data for use in displaying diagrams, each diagram comprising a plurality of entities, said apparatus comprising:

means for entering data for use in displaying diagrams;

means for storing data describing an individual diagram entity as a software object having a set of attributes which includes an identifier for the software object;

means for storing general data relating to an individual diagram as a software object whose attributes include an identifier for the software object;

means for storing data relating to the location of an individual entity on a diagram as a software object whose attributes include the location of the entity on the diagram and a pointer to the software object which describes the entity;

means for retrieving data describing individual entities of a diagram; and

means for using the retrieved data for displaying a diagram.

2. An apparatus as in claim 1, in which in the means for storing data relating to the location of an individual entity on a diagram as a software object, the attributes of the software object further include a pointer to the software object further include a pointer to the software object containing general data relating to the diagram.

3. An apparatus as in claim 1, in which the entities include graphical shapes and flow lines connecting the graphical shapes together.

4. An apparatus as in claim 3, in which data describing an entity in the form of a graphical shape is stored as a software object belonging to a software object class for graphical shapes, and data describing an entity in the form of a flow line is stored as a software object belonging to a software object class for flow lines whose attributes include a pair of pointers which point to the identifiers of the two software objects which describe, respectively, the entity at one end and the entity at the other end of the flow line.

5. An apparatus as in claim 4, including:

first means for finding a software object belonging to a software object class for graphical shapes which describes a graphical shape selected by a user of the apparatus;

second means for finding each software object belonging to the software object class which represents a flow line having one end connected to the selected graphical shape; and

third means for finding the software objects belonging to the software object class for graphical shapes which describe the graphical shapes connected to the other ends of said flow lines;

said data displaying means being arranged to use the data contained in the software objects found by said first, second and third software object finding means to display the selected graphical shape together with the flow lines having one end connected to the selected graphical shape and the graphical shapes connected to the other end of the flow lines.

6. An apparatus as in claim 3, in which the attributes of each software object which represents a graphical shape include a period of validity for the graphical shape.

7. An apparatus as in claim 6, including:

first means for finding software objects representing entities which together form a first diagram selected by a user of the apparatus, said first diagram representing a particular physical or abstract arrangement on a first date;

second means for finding software objects representing entities which together form a second diagram selected by the user, said second diagram representing said particular physical or abstract arrangement on a second date; and

means for selecting those software objects from the software objects found by the first and second software object finding means which represent entities valid on a date specified by the user, said specified data falling between said first and second dates; and

said data displaying means being arranged to use the data contained in software objects selected by the software object selecting means to display a diagram containing the entities which are valid on the specified date.

8. An apparatus as in claim 1, including means for operating on the retrieved data in a predetermined manner to produce a desired diagram.

9. An apparatus as in claim 1 including:

first means for finding software objects representing entities which together form a first diagram selected by a user of the apparatus;

second means for finding software objects representing entities which together form a second diagram selected by the user;

third means for finding software objects representing entities which are present in both the first and second diagrams;

fourth means for finding software objects representing entities present only in the first diagram; and

fifth means for finding software objects representing entities present only in the second diagram;

said data displaying means being arranged to display entities present in both the first and second diagrams in a first manner, entities present only in the first diagram in a second manner, and entities present only in the second diagram in a third manner.

10. An apparatus as in claim 1, in which for at least some of the software objects, the attributes of the software object include the security level required by a user of the apparatus to view a diagram including the entity represented by the software object.

11. An apparatus as in claim 1, in which said data storing means is arranged to partition the stored data into a set of scenarios arranged in hierarchical manner, said apparatus further including:

means for copying data from a higher level scenario to a lower level scenario; and means for promoting data from a lower level scenario to a higher level scenario.

12. An apparatus as in claim 11, including means for reusing data from a higher level scenario in a lower level scenario without copying the data to the lower level scenario.

13. An apparatus as in claim 1 further including means for generating a diagram containing graphical shapes which represent components of a real world structure in one domain from a diagram containing graphical shapes which represent the same real world structure in another domain.

14. A computer apparatus including a central processing unit, a memory, means for entering data and a display unit, said memory containing a program for controlling the computer apparatus and which is arranged to:

receive data for use in displaying diagrams;

store data representing an individual diagram entity as a software object having a set of attributes which includes an identifier for the software object;

store general data relating to an individual diagram as a software object whose attributes include an identifier for the software object;

store data relating to the location of an individual entity on a diagram as a software object whose attributes include the location of the entity on the diagram and a pointer to the software object which describes the entity;

retrieve data describing individual diagram entities of a diagram; and

use the retrieval data for displaying a diagram.

15. A method of operating a computer apparatus for storing diagram data and using the stored data for displaying diagrams, each diagram comprising a plurality of entities, said method comprising the steps of:

storing data describing each individual diagram entity as a software object having a set of attributes which includes an identifier for the object;

storing general data relating to each individual diagram as a software object whose attributes include an identifier for the software object;

storing data relating to the location of each individual entity of a diagram as a software object whose attributes include the location of the entity on the diagram and a pointer to the software object which describes the entity;

retrieving data describing individual entities of a diagram; and

using the retrieved data for displaying a diagram.

16. A method as in claim 15 in which in said step of storing data relating to the location of each individual entity of a diagram as a software object, the attributes of the software object include a pointer to the software object containing general data relating to the diagram.

17. A method as in claim 15, in which the entities include graphical shapes and flow lines connecting the graphical shapes together.

18. A method as in claim 17, in which data describing an entity in the form of a graphical shape is stored as a software object belonging to a software object class for graphical shapes and data describing an entity in the form of a flow line is stored as a software object belonging to a software object class for flow lines whose attributes include a pair of pointers which point to the identifier of the two software objects which describe, respectively, the entity at one end and the entity at the other end of the flow line.

19. A method as in claim 15, including the step of operating on the retrieved data in a predetermined manner to produce a desired diagram.

* * * * *

US005754738A

United States Patent [19]

Saucedo et al.

[11] Patent Number: 5,754,738

[45] Date of Patent: May 19, 1998

[54] COMPUTERIZED PROTOTYPING SYSTEM EMPLOYING VIRTUAL SYSTEM DESIGN ENVIROMENT

[75] Inventors: Robert Saucedo, Palos Verdos Estates, Calif.; Kootala P. Venugopal, Greenbelt, Md.

[73] Assignee: CAMC Corporation, Torrance, Calif.

[21] Appl. No.: 660,379

[22] Filed: Jun. 7, 1996

[51] Int. Cl.^{6} G06F 15/18; G06F 17/20

[52] U.S. Cl. 395/12; 395/10; 395/22; 395/50

[58] Field of Search 395/22, 23, 229, 395/500, 10–12, 50–51; 364/468.03, 468, 148

[56] References Cited

U.S. PATENT DOCUMENTS

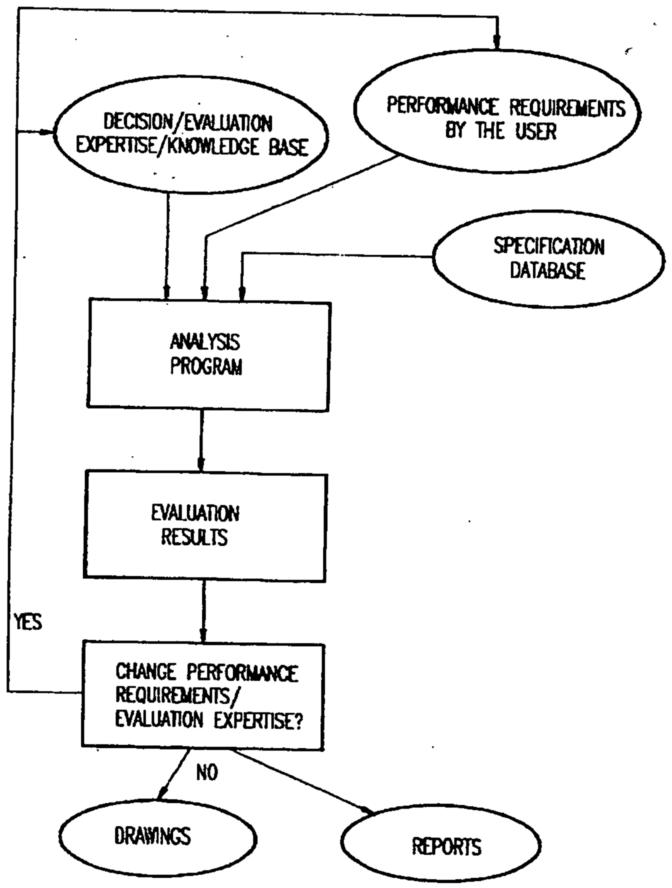
4,875,162	10/1989	Ferriter et al.	364/401
4,965,741	10/1990	Winchell et al.	364/513
5,283,857	2/1994	Simoudis	364/77
5,297,054	3/1994	Kienzle et al.	364/474.24
5,297,057	3/1994	Kramer et al.	364/512
5,333,237	7/1994	Stefanopoulos et al.	395/12
5,446,652	8/1995	Peterson et al.	364/578
5,450,527	9/1995	Wang	395/22
5,552,995	9/1996	Sebastian	364/468.03
5,555,406	9/1996	Nozawa	395/500

Primary Examiner—Tariq R. Hafiz
Attorney, Agent, or Firm—Jones & Volentine, L.L.P.

[57] ABSTRACT

In a computer prototyping system, a design is identified from a collection of alternative designs, the identified design best satisfying a set of conceptual level design specifications. One of these alternative designs is selected and their characteristic optimized based on the conceptual level design specifications. The specifications are modified interactively using graphical interfaces for re-evaluating and re-optimizing the designs. Simulation of functional as well as geometrical properties of the design and its components are effected on a computer using graphical design browsers. The performance is analyzed against a set of design specification and the user is allowed to interactively redesign by selecting one of the previous design operations through the graphical interfaces.

7 Claims, 26 Drawing Sheets


DECISION/EVALUATION EXPERTISE/KNOWLEDGE BASE
PERFORMANCE REQUIREMENTS BY THE USER
SPECIFICATION DATABASE
ANALYSIS PROGRAM
EVALUATION RESULTS
YES
CHANGE PERFORMANCE REQUIREMENTS/ EVALUATION EXPERTISE?
NO
DRAWINGS
REPORTS

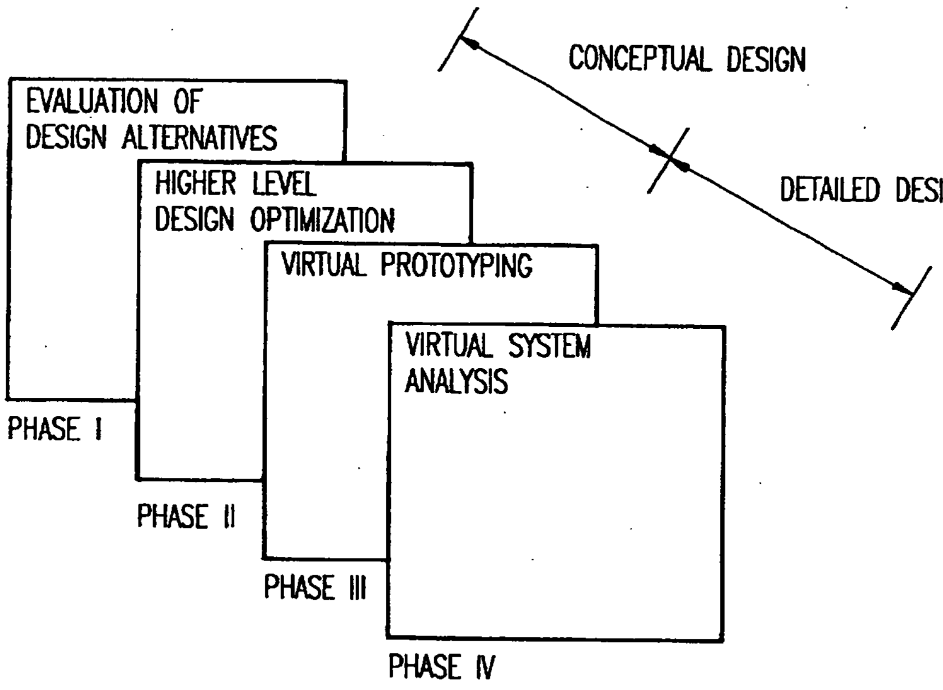

CONCEPTUAL DESIGN
DETAILED DESIGN
EVALUATION OF DESIGN ALTERNATIVES
HIGHER LEVEL DESIGN OPTIMIZATION
VIRTUAL PROTOTYPING
VIRTUAL SYSTEM ANALYSIS
PHASE I
PHASE II
PHASE III
PHASE IV

FIG.1

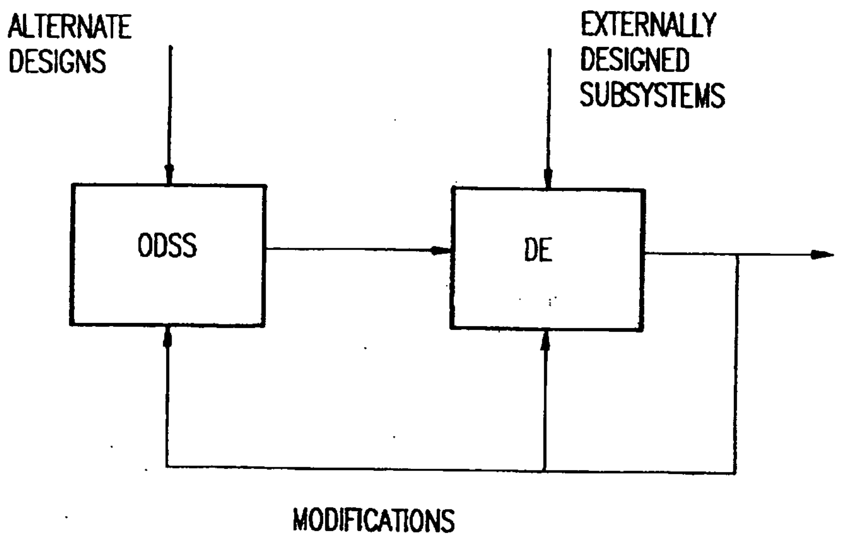

ALTERNATE DESIGNS
EXTERNALLY DESIGNED SUBSYSTEMS
ODSS
DE
MODIFICATIONS

FIG.2

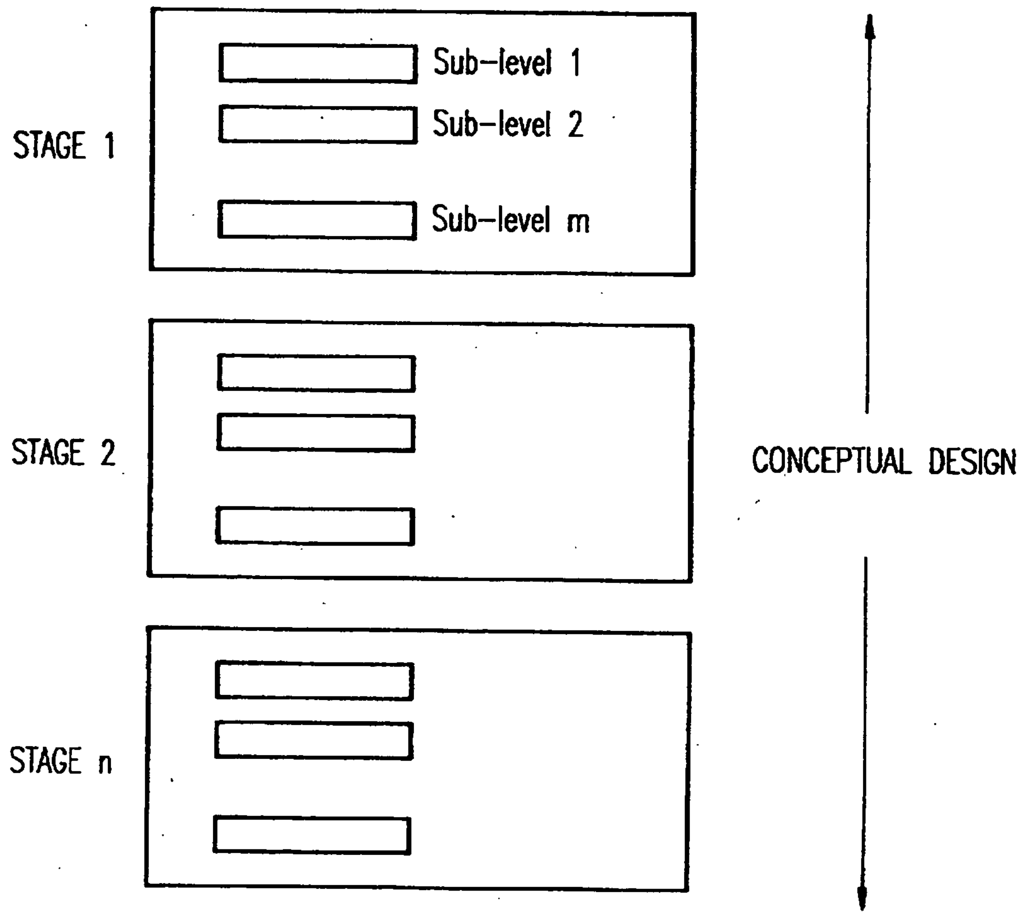

STAGE 1
Sub-level 1
Sub-level 2
Sub-level m
STAGE 2
STAGE n
CONCEPTUAL DESIGN

FIG.3

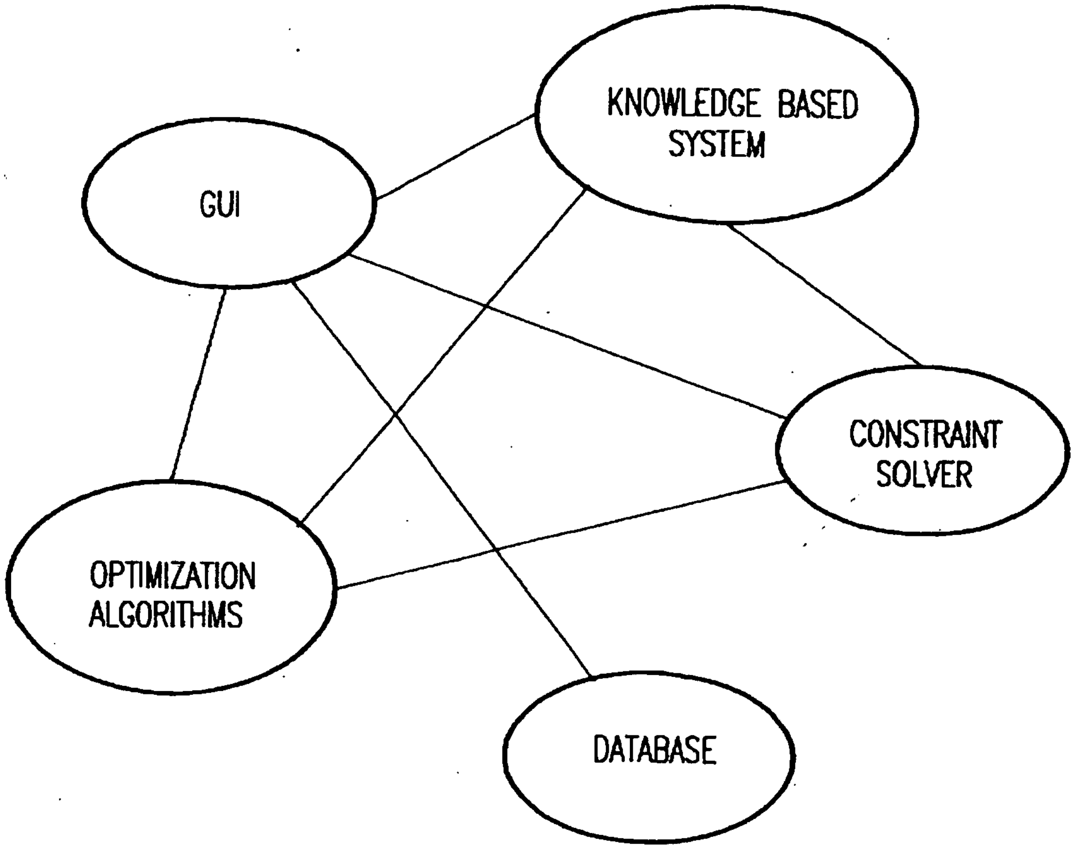

KNOWLEDGE BASED SYSTEM
GUI
CONSTRAINT SOLVER
OPTIMIZATION ALGORITHMS
DATABASE

FIG.4

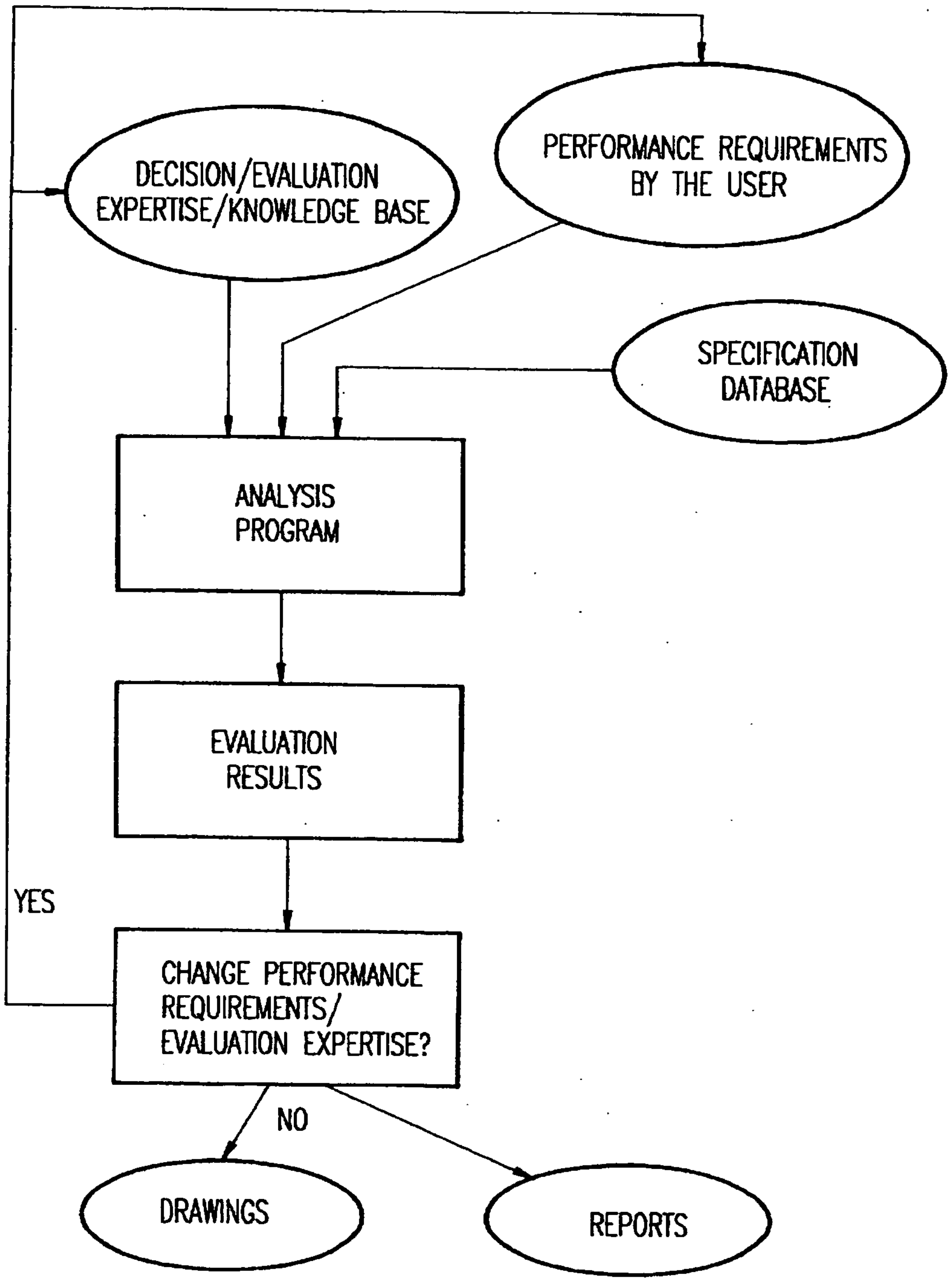

PERFORMANCE REQUIREMENTS BY THE USER
DECISION/EVALUATION EXPERTISE/KNOWLEDGE BASE
SPECIFICATION DATABASE
ANALYSIS PROGRAM
EVALUATION RESULTS
YES
CHANGE PERFORMANCE REQUIREMENTS/ EVALUATION EXPERTISE?
NO
DRAWINGS
REPORTS

FIG.5

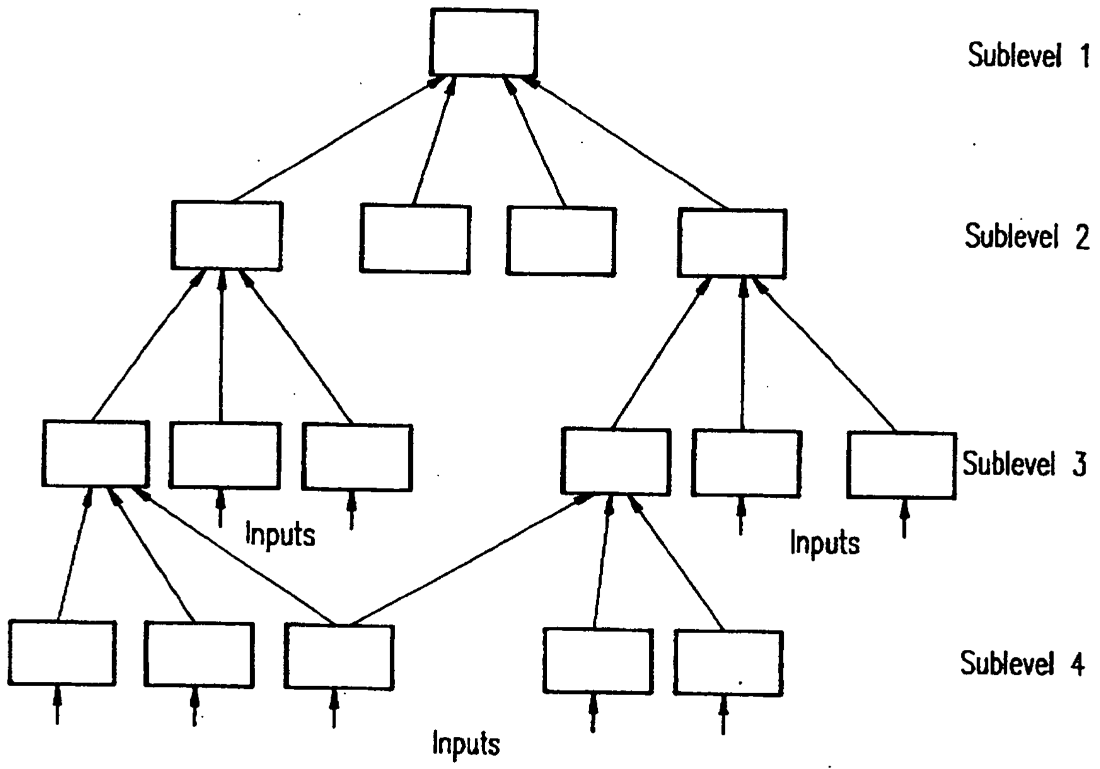

Sublevel 1
Sublevel 2
Sublevel 3
Sublevel 4
Inputs
Inputs
Inputs

FIG.6

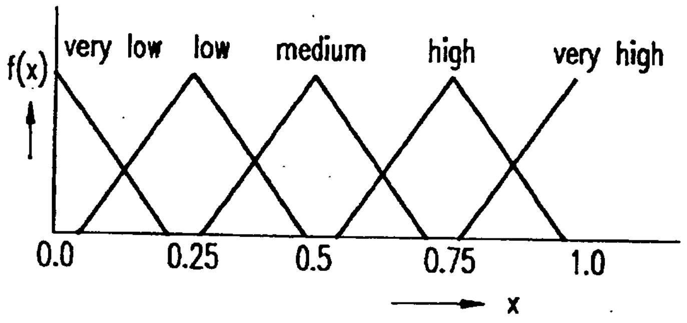

f(x)
very low
low
medium
high
very high
0.0
0.25
0.5
0.75
1.0
x

FIG.7

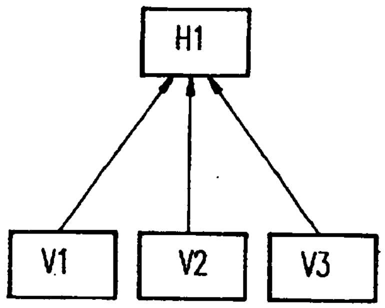


FIG.8

	V1	V2	V3
V1	1	a	b
V2	1/a	1	c
V3	1/b	1/c	1

FIG.9

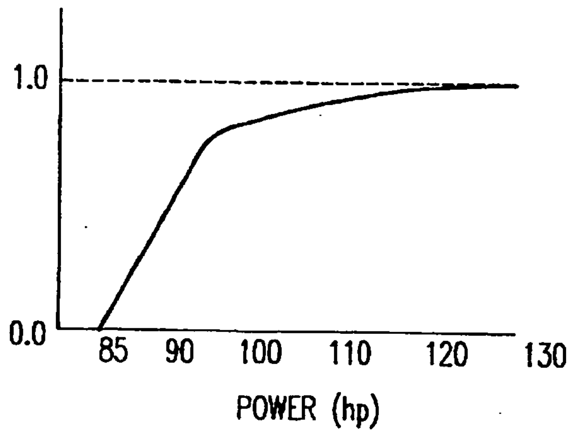


FIG.10A

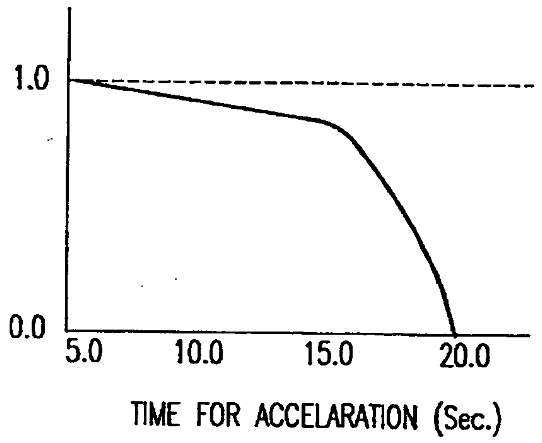
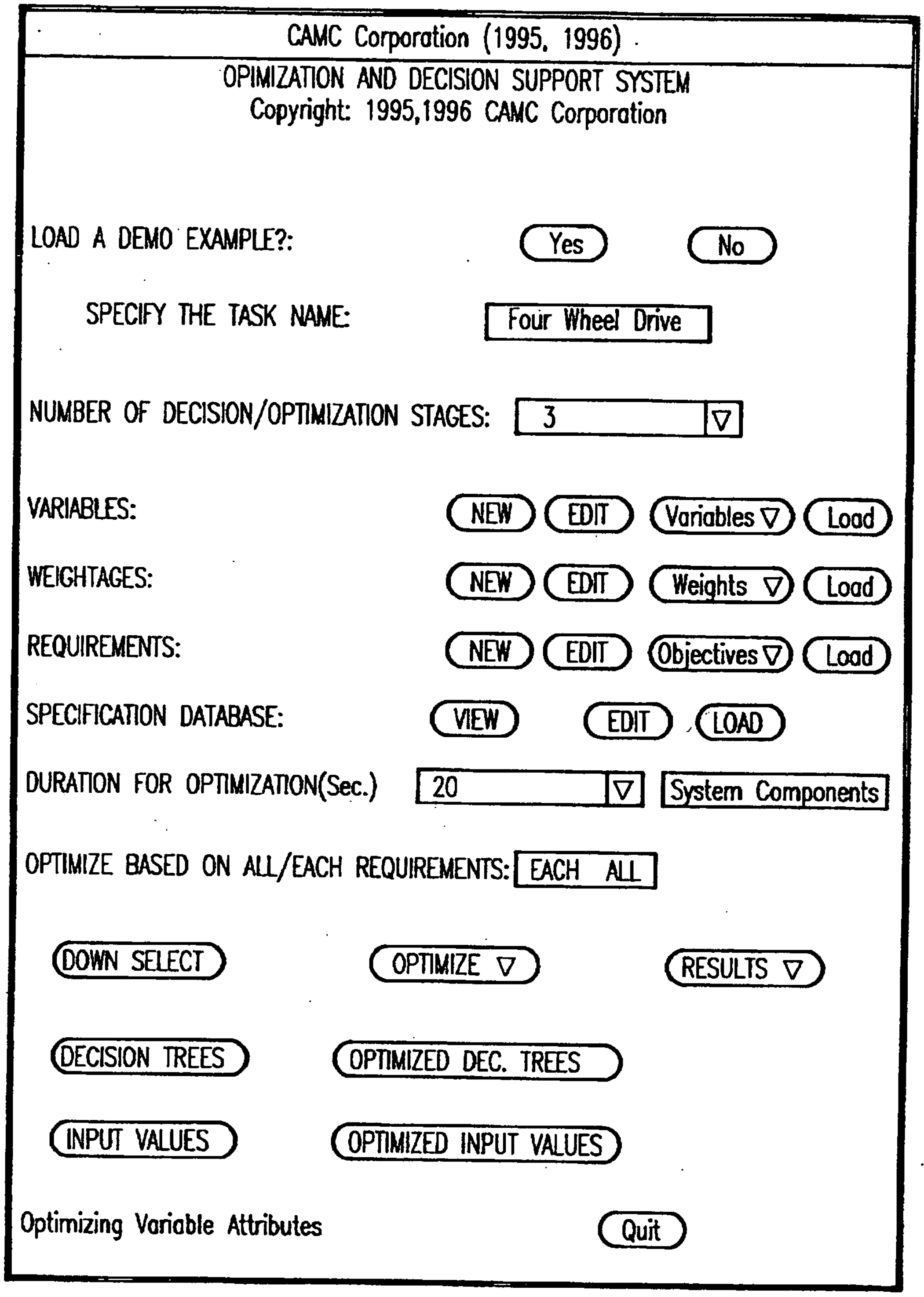


FIG.10B


CAMC Corporation (1995, 1996)

OPIMIZATION AND DECISION SUPPORT SYSTEM
Copyright: 1995,1996 CAMC Corporation

LOAD A DEMO EXAMPLE?: Yes No

SPECIFY THE TASK NAME: Four Wheel Drive

NUMBER OF DECISION/OPTIMIZATION STAGES: 3 ▽

VARIABLES: NEW EDIT Variables ▽ Load

WEIGHTAGES: NEW EDIT Weights ▽ Load

REQUIREMENTS: NEW EDIT Objectives ▽ Load

SPECIFICATION DATABASE: VIEW EDIT LOAD

DURATION FOR OPTIMIZATION(Sec.) 20 ▽ System Components

OPTIMIZE BASED ON ALL/EACH REQUIREMENTS: EACH ALL

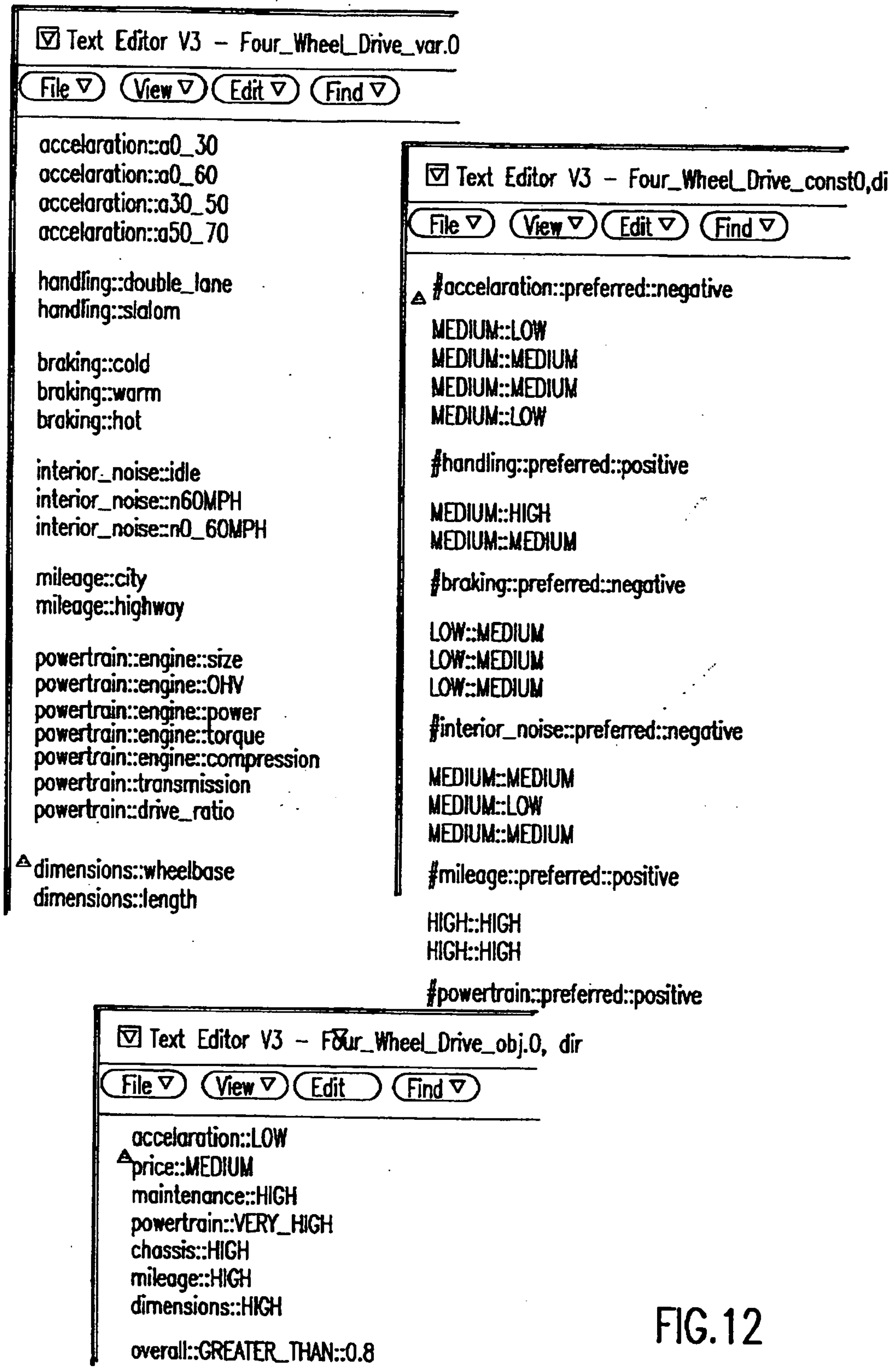
DOWN SELECT OPTIMIZE ▽ RESULTS ▽

DECISION TREES OPTIMIZED DEC. TREES

INPUT VALUES OPTIMIZED INPUT VALUES

Optimizing Variable Attributes Quit

FIG.11

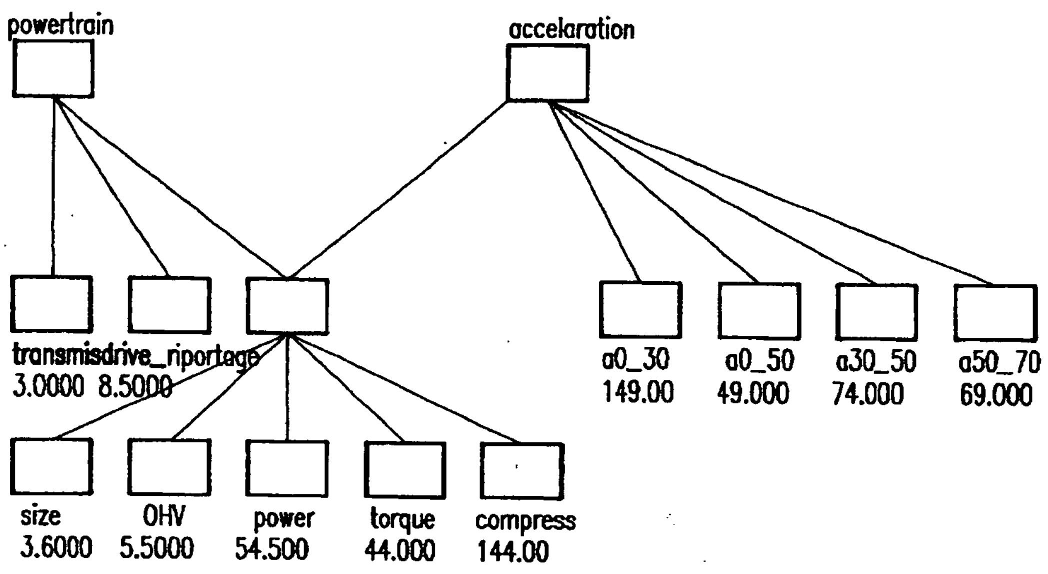


```
accelaration::a0_30
accelaration::a0_60
accelaration::a30_50
accelaration::a50_70

handling::double_lane
handling::slalom

braking::cold
braking::warm
braking::hot

interior_noise::idle
interior_noise::n60MPH
interior_noise::n0_60MPH

mileage::city
mileage::highway

powertrain::engine::size
powertrain::engine::OHV
powertrain::engine::power
powertrain::engine::torque
powertrain::engine::compression
powertrain::transmission
powertrain::drive_ratio

dimensions::wheelbase
dimensions::length
```

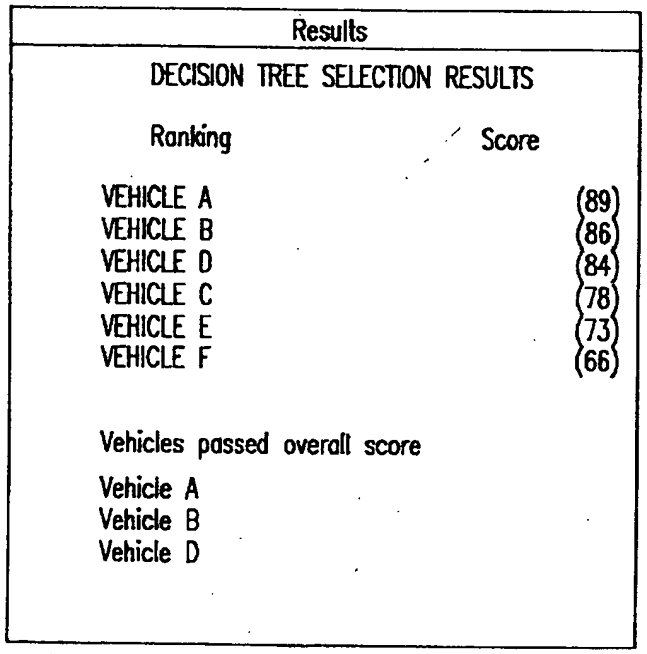
```
#accelaration::preferred::negative

MEDIUM::LOW
MEDIUM::MEDIUM
MEDIUM::MEDIUM
MEDIUM::LOW

#handling::preferred::positive

MEDIUM::HIGH
MEDIUM::MEDIUM

#braking::preferred::negative

LOW::MEDIUM
LOW::MEDIUM
LOW::MEDIUM

#interior_noise::preferred::negative

MEDIUM::MEDIUM
MEDIUM::LOW
MEDIUM::MEDIUM

#mileage::preferred::positive

HIGH::HIGH
HIGH::HIGH

#powertrain::preferred::positive
```

```
accelaration::LOW
price::MEDIUM
maintenance::HIGH
powertrain::VERY_HIGH
chassis::HIGH
mileage::HIGH
dimensions::HIGH

overall::GREATER_THAN::0.8
```

FIG.12



VEHICLE_A


Results

DECISION TREE SELECTION RESULTS

Ranking	Score
VEHICLE A	(89)
VEHICLE B	(86)
VEHICLE D	(84)
VEHICLE C	(78)
VEHICLE E	(73)
VEHICLE F	(66)

Vehicles passed overall score

Vehicle A
Vehicle B
Vehicle D

FIG.13

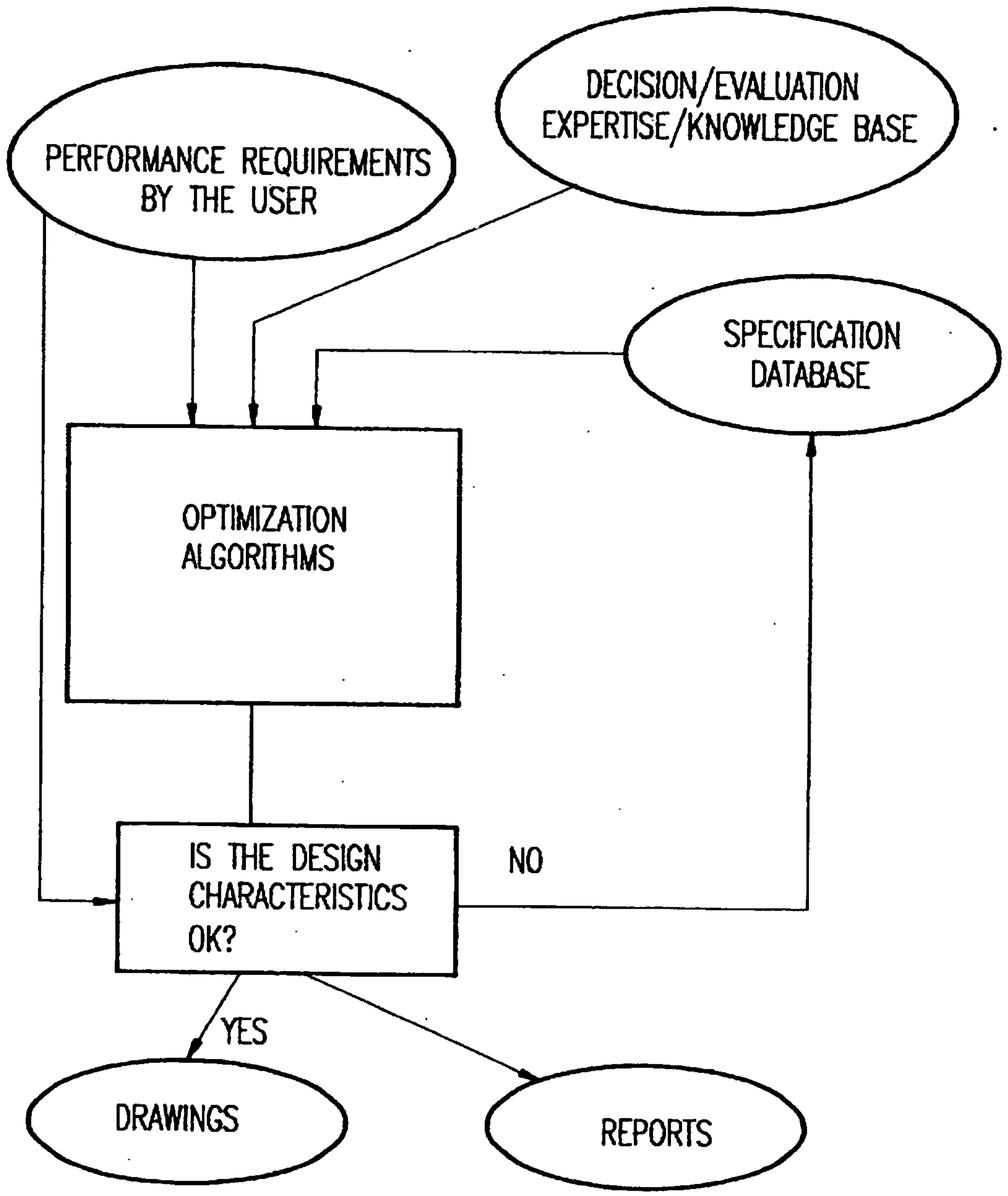

PERFORMANCE REQUIREMENTS BY THE USER
DECISION/EVALUATION EXPERTISE/KNOWLEDGE BASE
SPECIFICATION DATABASE
OPTIMIZATION ALGORITHMS
IS THE DESIGN CHARACTERISTICS OK?
NO
YES
DRAWINGS
REPORTS

FIG.14

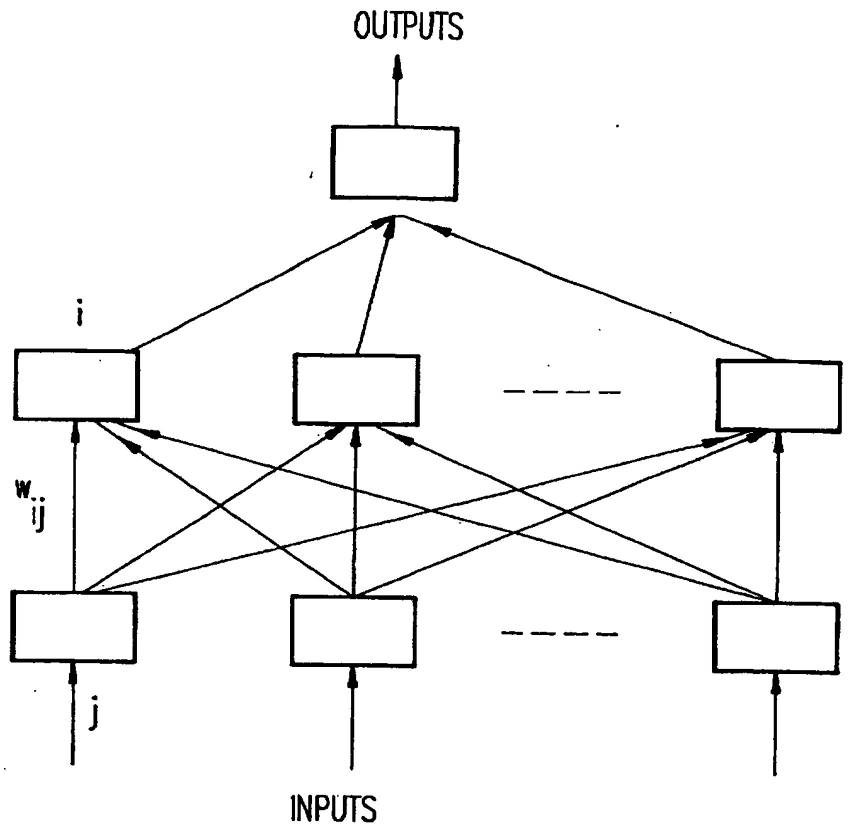


FIG.15

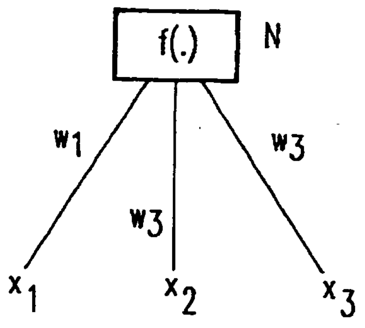
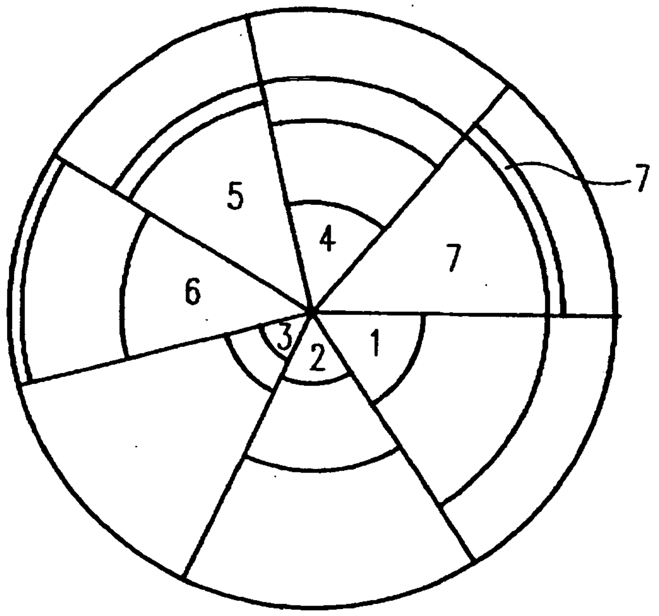


FIG.16



1	ACCELARATION
2	PRICE
3	MAINTENANCE
4	POWERTRAIN
5	CHASSIS
6	MILEAGE
7	DIMENSIONS

FIG.17

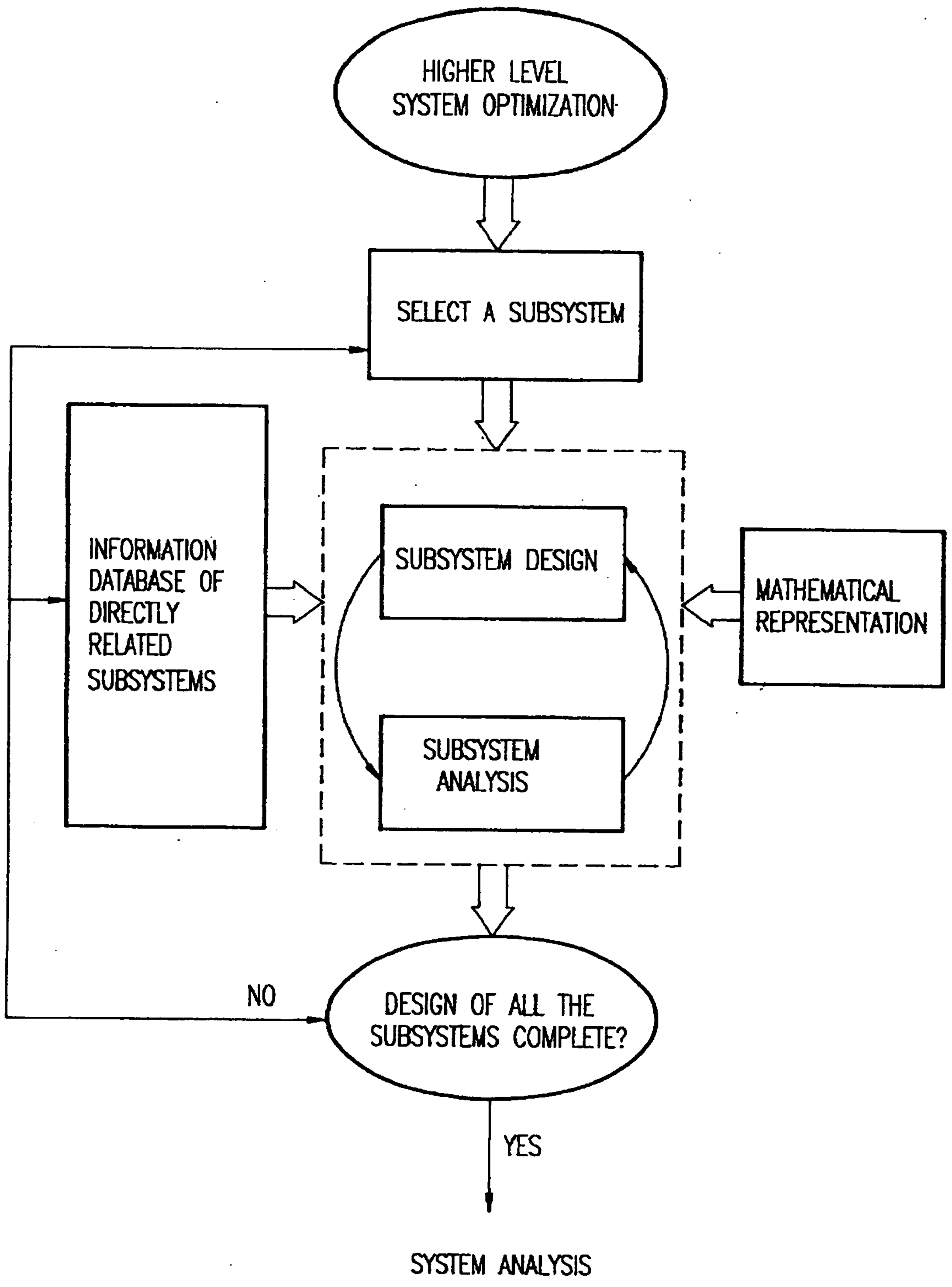

HIGHER LEVEL SYSTEM OPTIMIZATION
SELECT A SUBSYSTEM
INFORMATION DATABASE OF DIRECTLY RELATED SUBSYSTEMS
SUBSYSTEM DESIGN
SUBSYSTEM ANALYSIS
MATHEMATICAL REPRESENTATION
NO
DESIGN OF ALL THE SUBSYSTEMS COMPLETE?
YES
SYSTEM ANALYSIS

FIG.18

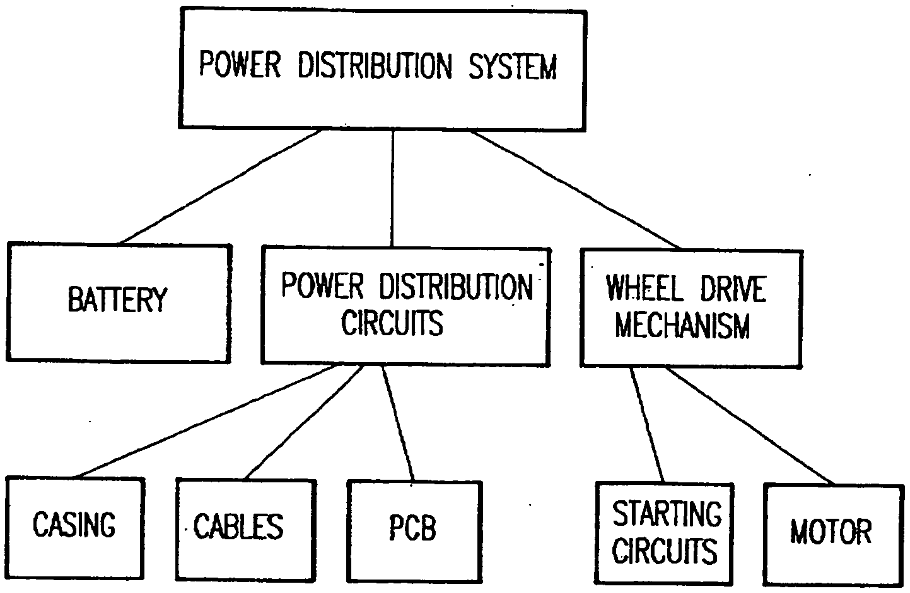


FIG.19

	BATTERY	POWER DISTRIB. CIRCUIT	WHEEL DRIVE
BATTERY	X	MEDIUM	HIGH
POWER DISTRIB. CIRCUIT	VERY LOW	X	MEDIUM
WHEEL DRIVE	LOW	LOW	X

FIG.20

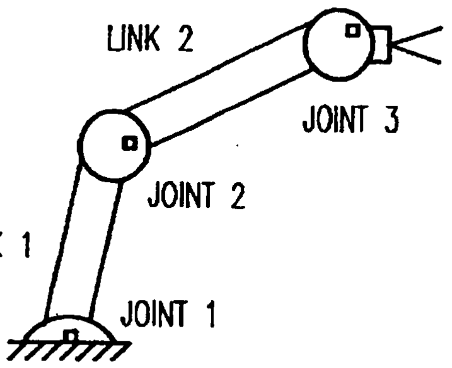

LINK 2
JOINT 3
JOINT 2
LINK 1
JOINT 1

FIG.21

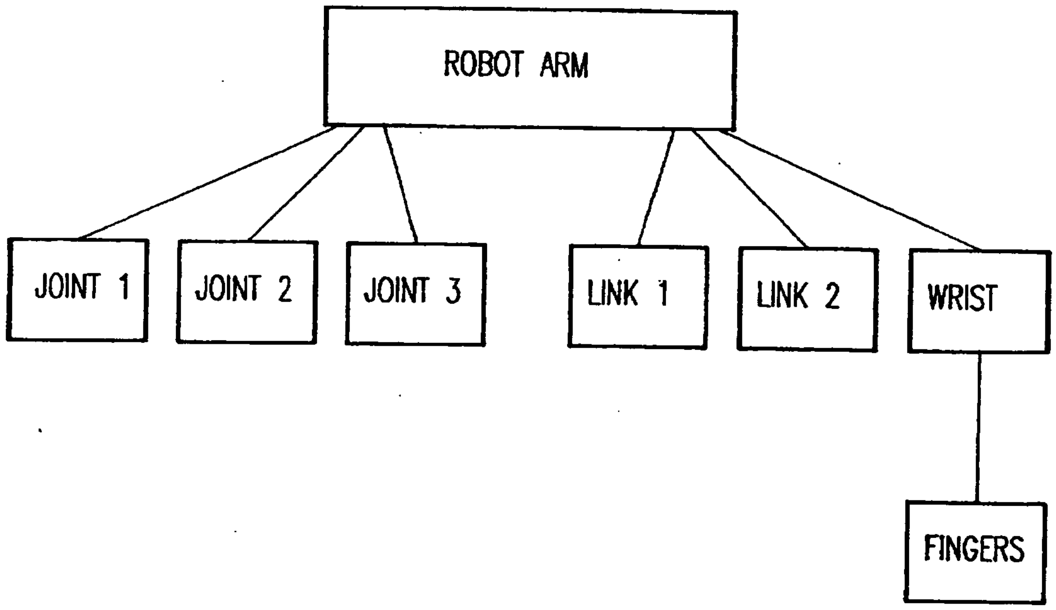

ROBOT ARM
JOINT 1
JOINT 2
JOINT 3
LINK 1
LINK 2
WRIST
FINGERS

FIG.22

	JOINT 1	JOINT 2	JOINT 3	LINK 1	LINK 2	WRIST
JOINT 2		0	0	1	0	0
JOINT 3	0		0	1	1	0
LINK 1	0	0		0	1	1
LINK 2	1	1	0		0	0
WRIST	0	1	1			0
JOINT 1	0	0	1	0	0	

FIG.23

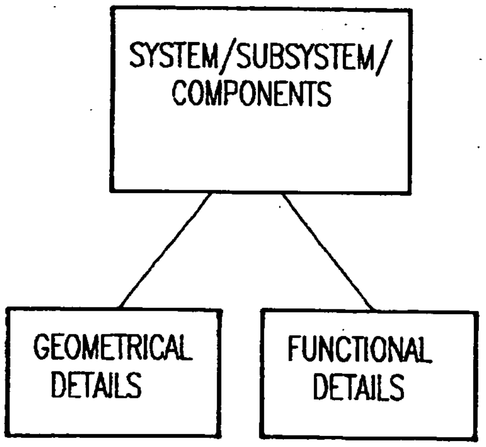


FIG.24A

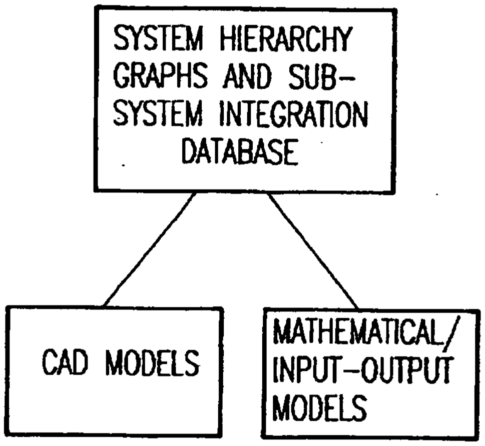

SYSTEM HIERARCHY GRAPHS AND SUB-SYSTEM INTEGRATION DATABASE
CAD MODELS
MATHEMATICAL/ INPUT-OUTPUT MODELS

FIG.24B

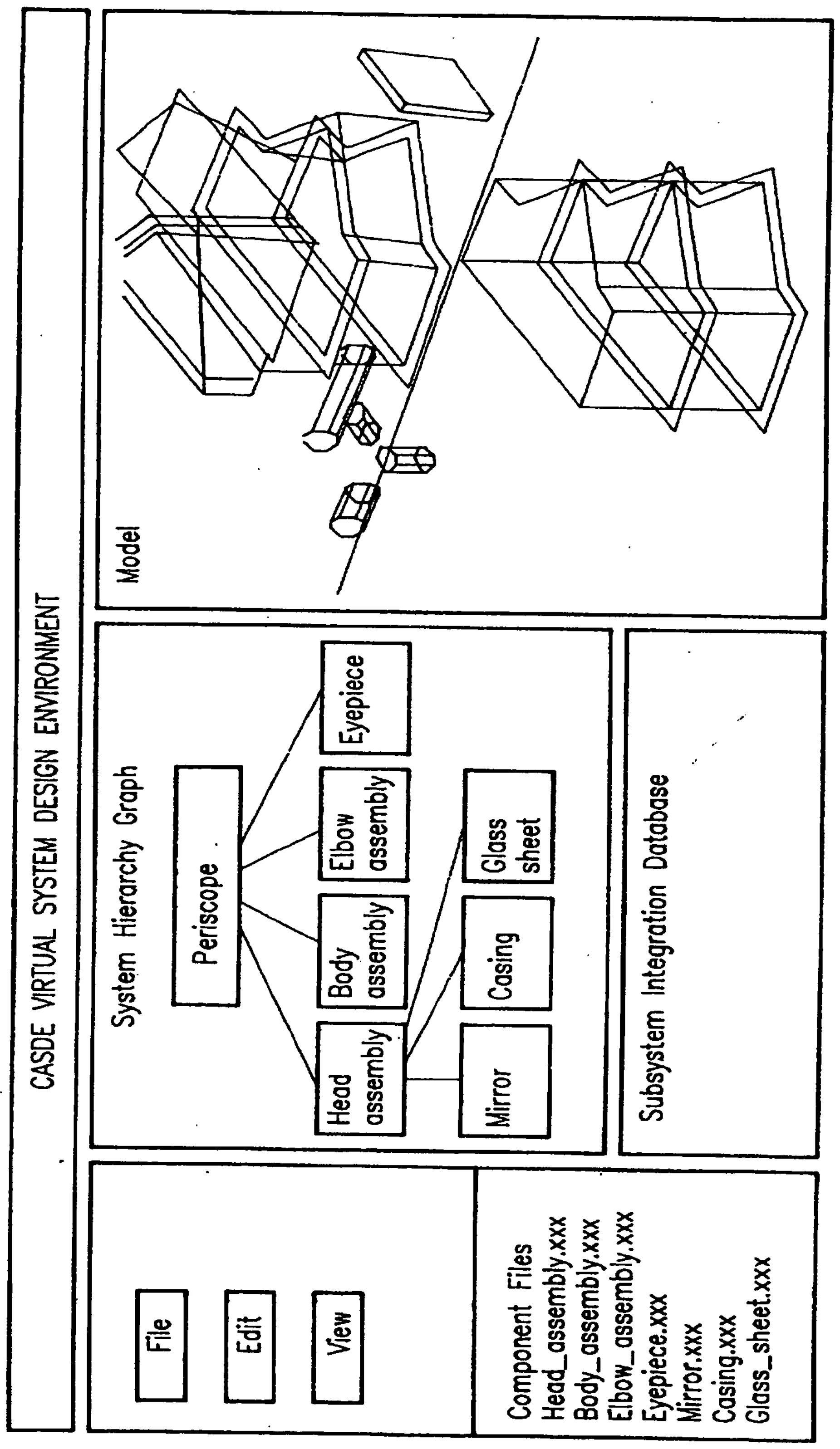

CASDE VIRTUAL SYSTEM DESIGN ENVIRONMENT
Model
System Hierarchy Graph
Periscope
Head assembly
Body assembly
Elbow assembly
Eyepiece
Mirror
Casing
Glass sheet
Subsystem Integration Database
File
Edit
View
Component Files
Head_assembly.xxx
Body_assembly.xxx
Elbow_assembly.xxx
Eyepiece.xxx
Mirror.xxx
Casing.xxx
Glass_sheet.xxx

FIG.25

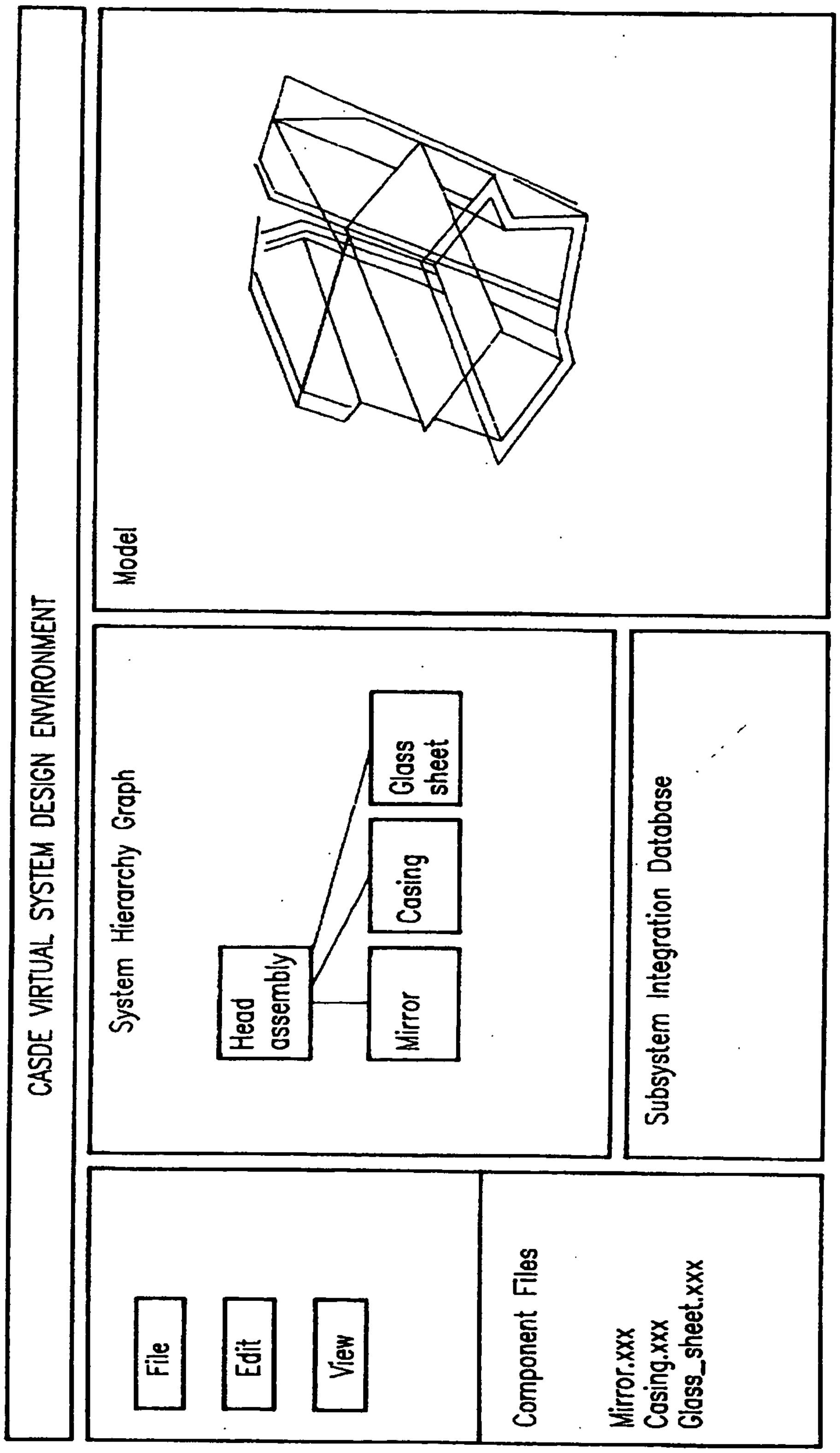

CASDE VIRTUAL SYSTEM DESIGN ENVIRONMENT
File
Edit
View
Component Files
Mirror.xxx
Casing.xxx
Glass_sheet.xxx
System Hierarchy Graph
Head assembly
Mirror
Casing
Glass sheet
Subsystem Integration Database
Model

FIG.26

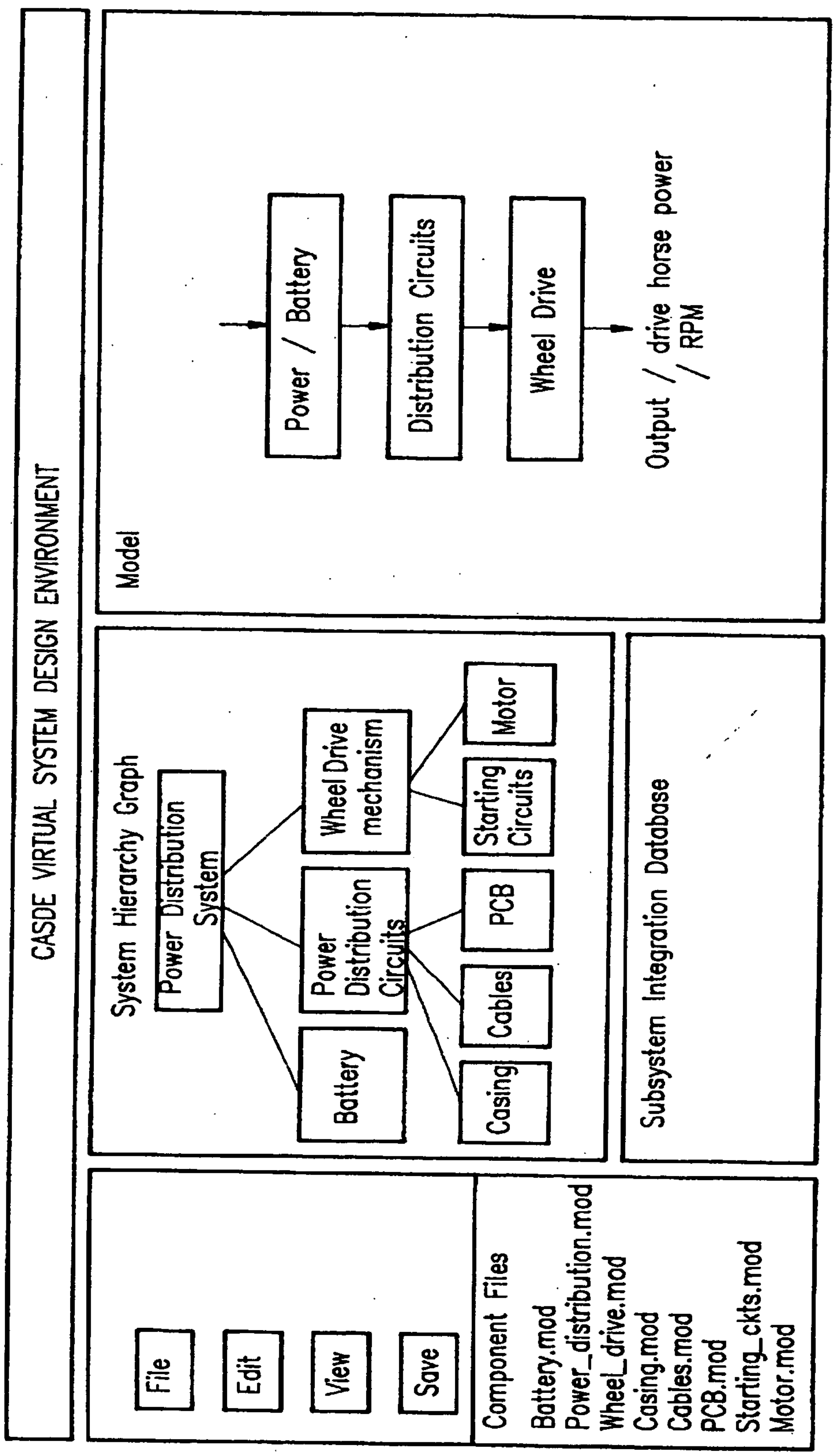

CASDE VIRTUAL SYSTEM DESIGN ENVIRONMENT
File
Edit
View
Save
Component Files
Battery.mod
Power_distribution.mod
Wheel_drive.mod
Casing.mod
Cables.mod
PCB.mod
Starting_ckts.mod
Motor.mod
System Hierarchy Graph
Power Distribution System
Battery
Power Distribution Circuits
Wheel Drive mechanism
Casing
Cables
PCB
Starting Circuits
Motor
Subsystem Integration Database
Model
Power / Battery
Distribution Circuits
Wheel Drive
Output / drive horse power / RPM

FIG.27

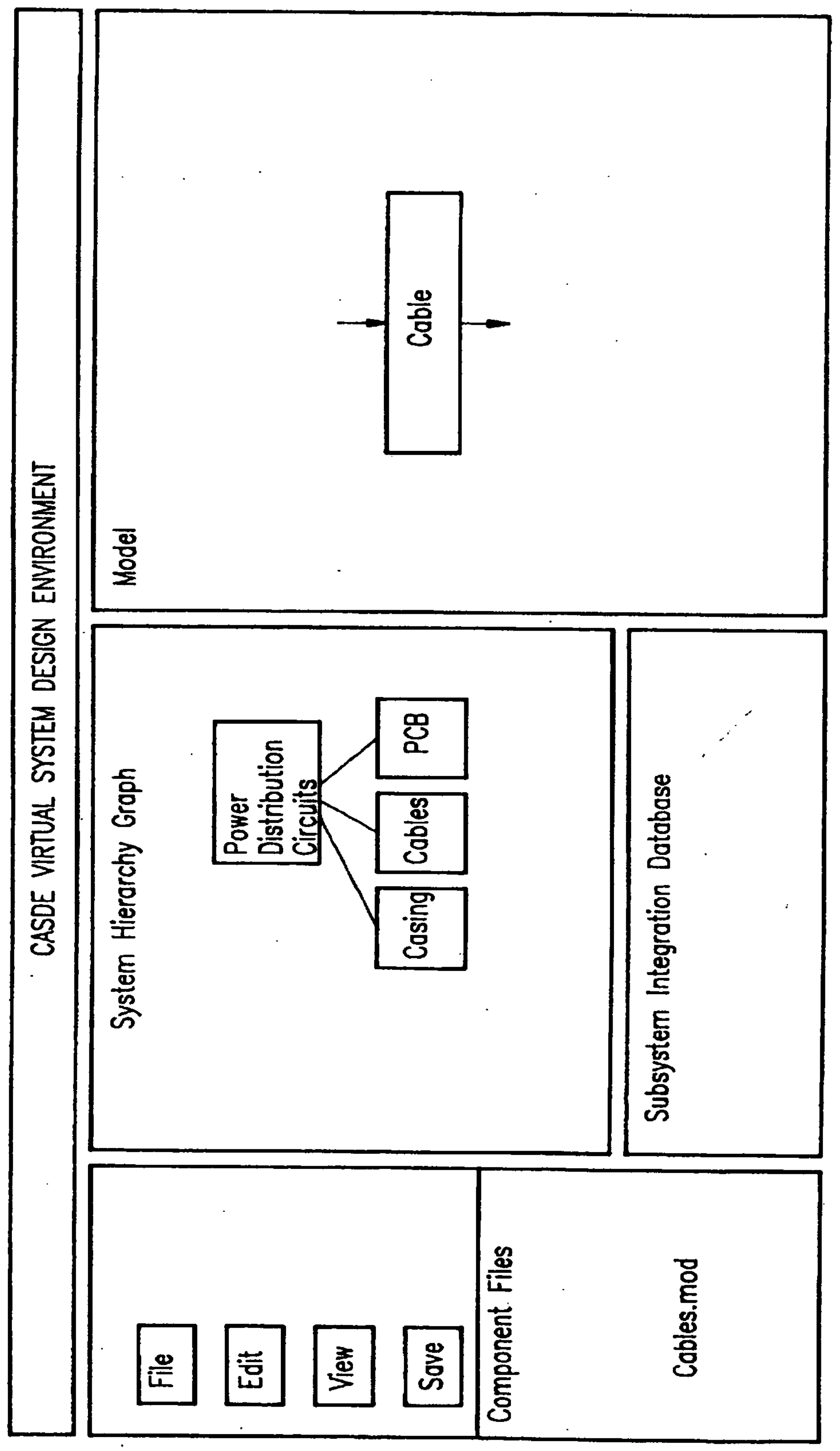

CASDE VIRTUAL SYSTEM DESIGN ENVIRONMENT
Model
Cable
System Hierarchy Graph
Power Distribution Circuits
PCB
Cables
Casing
Subsystem Integration Database
File
Edit
View
Save
Component Files
Cables.mod

FIG.28

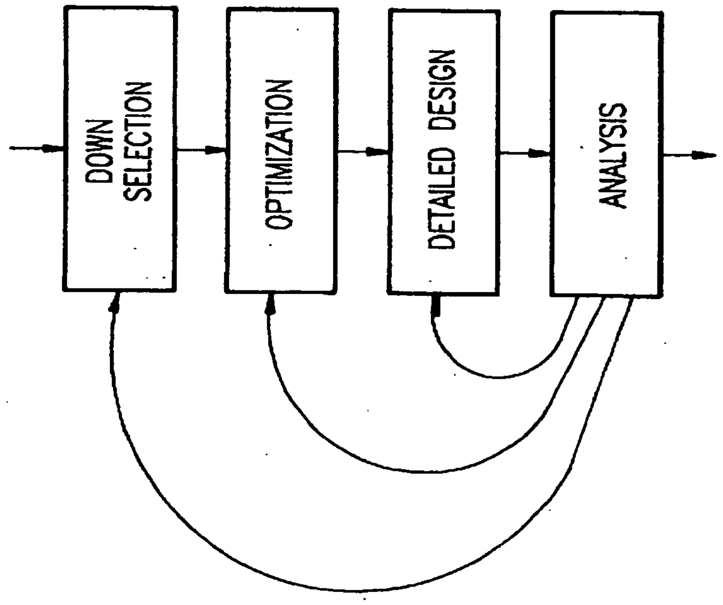


FIG.29

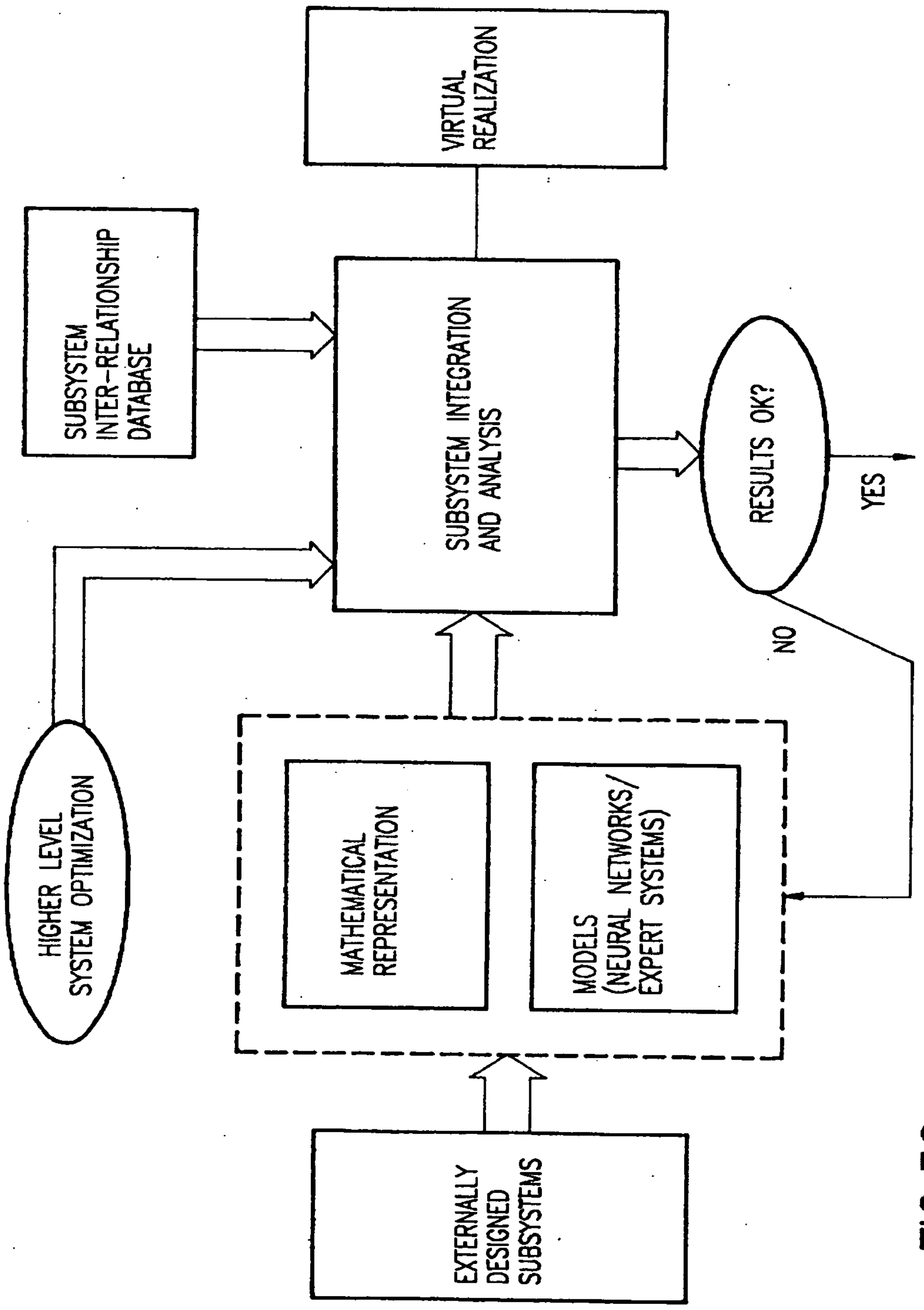

VIRTUAL REALIZATION
SUBSYSTEM INTER-RELATIONSHIP DATABASE
SUBSYSTEM INTEGRATION AND ANALYSIS
RESULTS OK?
YES
NO
HIGHER LEVEL SYSTEM OPTIMIZATION
MATHEMATICAL REPRESENTATION
MODELS (NEURAL NETWORKS/ EXPERT SYSTEMS)
EXTERNALLY DESIGNED SUBSYSTEMS

FIG.30

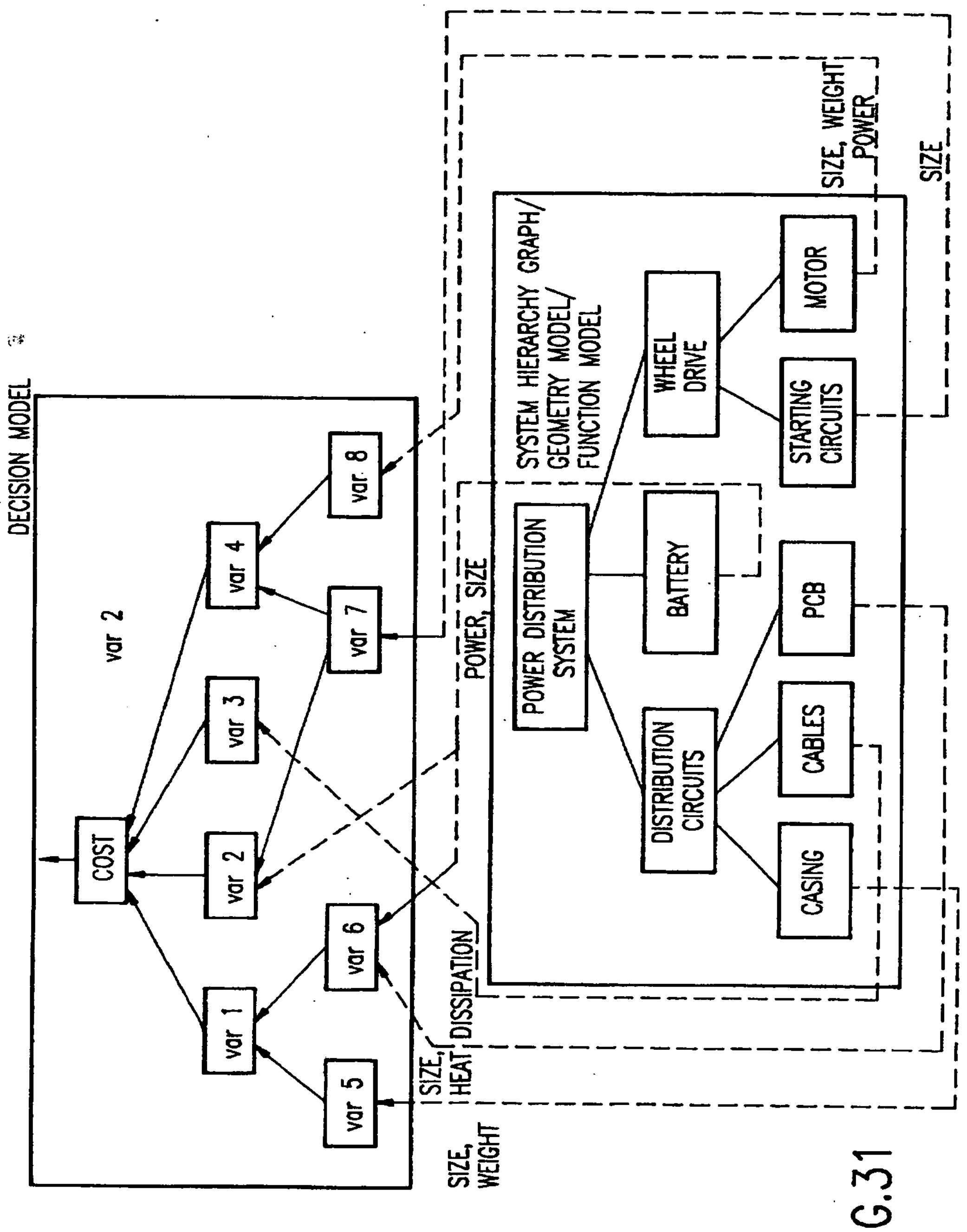

DECISION MODEL
var 2
COST
var 1
var 2
var 3
var 4
var 5
var 6
var 7
var 8
SYSTEM HIERARCHY GRAPH/
GEOMETRY MODEL/
FUNCTION MODEL
POWER DISTRIBUTION SYSTEM
BATTERY
WHEEL DRIVE
DISTRIBUTION CIRCUITS
PCB
CABLES
CASING
STARTING CIRCUITS
MOTOR
SIZE, WEIGHT
POWER
SIZE
POWER, SIZE
DISSIPATION
SIZE, HEAT
SIZE, WEIGHT

FIG.31

	var 2	var 3	var 5	var 6	var 7	var 8
Battery	1	0	0	1	0	0
Casing	0	0	1	0	0	0
PCB	0	0	0	1	0	0
Cables	0	1	0	0	0	0
Start. Ckts.	0	0	0	0	1	0
Motor	0	0	0	0	0	1

FIG.32

Motor

var 8

	Size	Weight	Power
Sub var 1	1	0	0
Sub var 2	0	0	1
Sub var 3	0	1	0

FIG.33A

PCB

var 6

	Size	Heat Dissip
Sub var 1	0	1
Sub var 2	1	0

FIG.33B

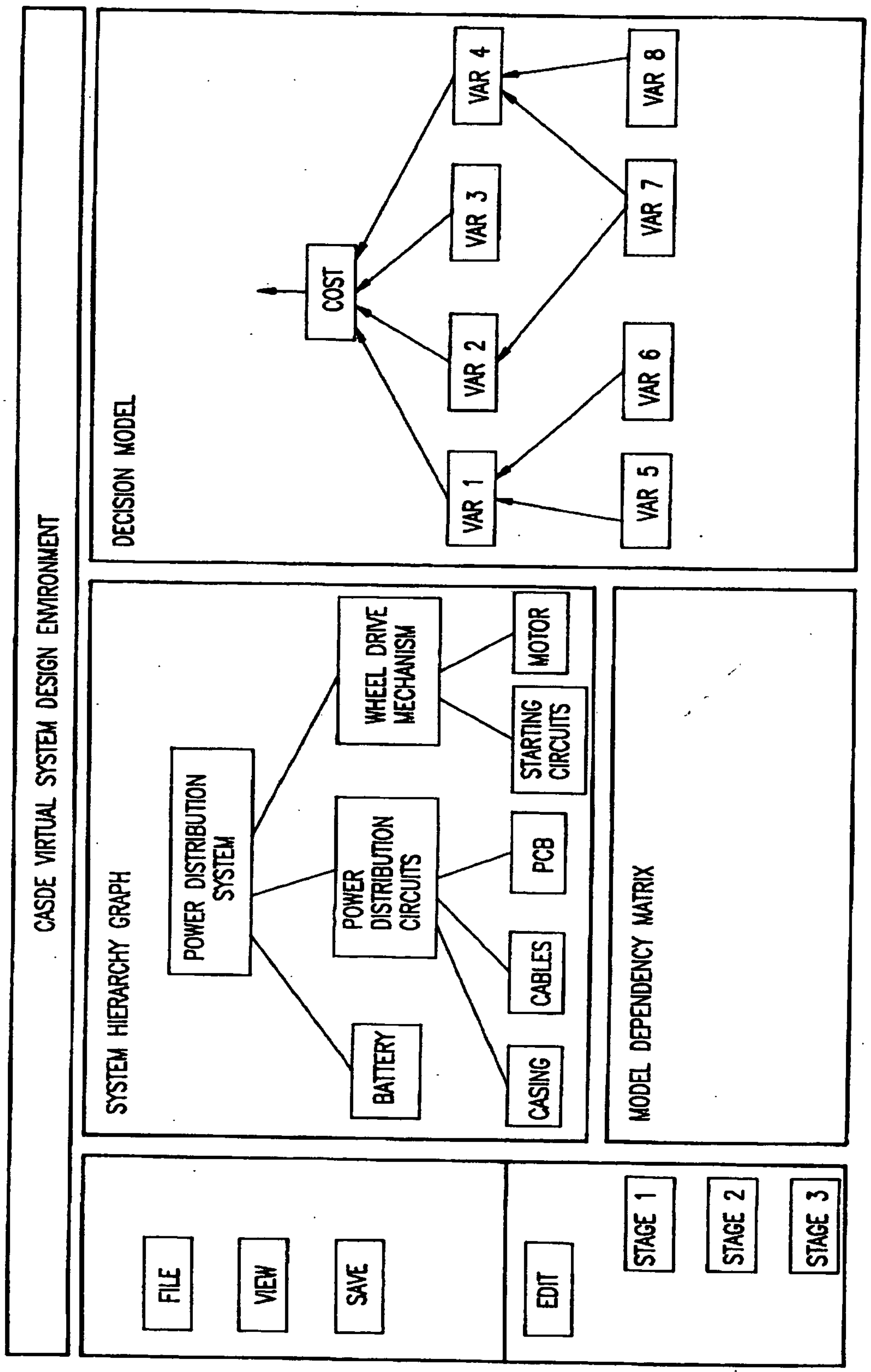

CASDE VIRTUAL SYSTEM DESIGN ENVIRONMENT
FILE
VIEW
SAVE
EDIT
STAGE 1
STAGE 2
STAGE 3
SYSTEM HIERARCHY GRAPH
POWER DISTRIBUTION SYSTEM
BATTERY
POWER DISTRIBUTION CIRCUITS
WHEEL DRIVE MECHANISM
CASING
CABLES
PCB
STARTING CIRCUITS
MOTOR
MODEL DEPENDENCY MATRIX
DECISION MODEL
COST
VAR 1
VAR 2
VAR 3
VAR 4
VAR 5
VAR 6
VAR 7
VAR 8

FIG.34

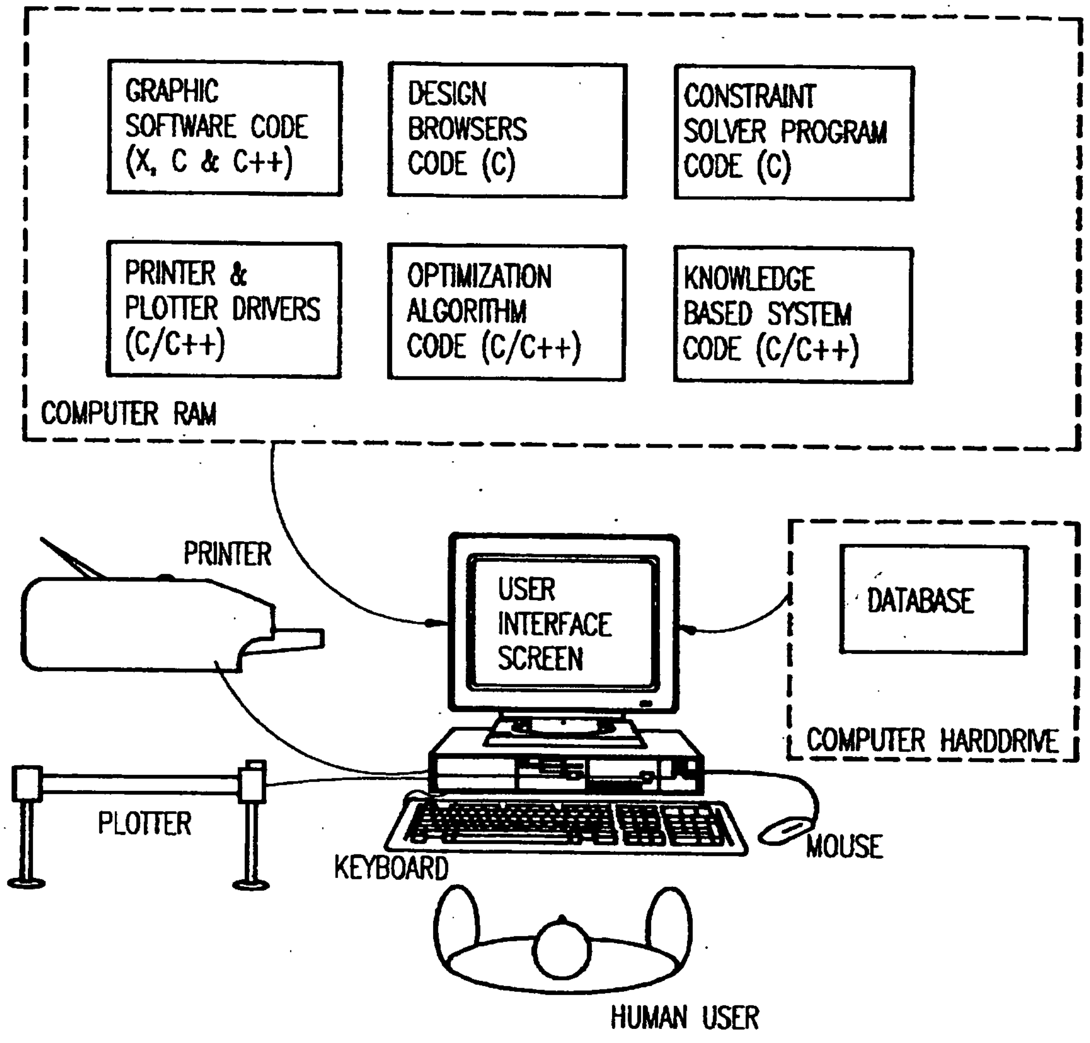

GRAPHIC SOFTWARE CODE (X, C & C++)
DESIGN BROWSERS CODE (C)
CONSTRAINT SOLVER PROGRAM CODE (C)
PRINTER & PLOTTER DRIVERS (C/C++)
OPTIMIZATION ALGORITHM CODE (C/C++)
KNOWLEDGE BASED SYSTEM CODE (C/C++)
COMPUTER RAM
PRINTER
USER INTERFACE SCREEN
DATABASE
COMPUTER HARDDRIVE
PLOTTER
KEYBOARD
MOUSE
HUMAN USER

FIG.35

COMPUTERIZED PROTOTYPING SYSTEM EMPLOYING VIRTUAL SYSTEM DESIGN ENVIROMENT

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is directed to a computerized prototyping system containing a virtual system design environment, and more particularly, to a computerized prototyping system which allows a user through a graphical user interface to dynamically change a model and to reevaluate its functions, and to automatically optimize the model with the help of a knowledged based expert system.

2. Description of the Related Art

The development of complex engineering systems initially requires levels of management and engineering decisions and trade-off studies. This design phase, normally called conceptual design phase, involves evaluation of a number of higher level performance and manufactorability requirements such as costs, profit, technical resources, market constraints, man power, and computing resources. These decisions in engineering designs are often done very heuristically, mainly based on the experience of the management and engineering design team.

It is estimated that about 80–90% of the product costs, engineering requirements and quality are decided at the first 20–25% of the design phase. This makes the conceptual design phase particularly important. Though there is a well established industry providing sophisticated computer aided Design (CAD) tools for detailed design of engineering systems, these tools are generally not suitable for the conceptual design phase.

SUMMARY OF THE INVENTION

The present invention is directed to a virtual system design environment (VSDE) which allows for an organized computer-implemented approach to conceptual design of complex systems. It provides a set of tools for systematically analyzing the decisions at the conceptual design phase, evaluation of candidate designs, higher level design optimization and virtual system prototyping. The application of VSDE is not restricted to the domain of engineering design analysis. It can be effectively used in a variety of applications such as management decision analysis, financial market evaluations, logistics, data mining and data interpretation.

The computer-implemented virtual system design environment solves the shortcomings of the current system design and analysis methodologies.

(1) The VSDE of the invention can be used for the entire design cycle of a system, starting from the conceptual design phase. It can easily analyze high level decision options against the performance requirements set forth by the designer, by adding, deleting or changing variables, associated constraints and design objectives. As mentioned previously, commercially available computer aided design (CAD) tools do not deal with conceptual design stage of system design.

(2) Prior knowledge about a decision process can be easily incorporated in the form of a knowledge based system (decision model). The designer can load an existing decision model or can create a new decision model during the conceptual design process. The decision model in the VSDE of the invention can be represented using fuzzy variables, allowing representation of decision uncertainties.

(3) The decision model can incorporate subsystems which are designed through the VSDE of the invention as well as external designs and analyze the overall system. Also, it allows optimization of the system parameters, incorporating these subsystems. For this optimization, exact mathematical representation of the subsystems need not be known. This aspect is a significant advantage over conventional Operations Research approach such as linear programming and nonlinear programming.

(4) Since the VSDE of the invention integrates all the stages in a typical system design, the design has complete flexibility in monitoring each stage in the design process, intermediate design parameters, and results of designs at each stage. This flexibility allows the designer to interrupt the process at any stage, modify the requirements or goals, or modify the variables and constraints.

(5) Any technological developments in subsystems can be readily incorporated into the VSDE of the invention, along with its constraints and interrelationships with other subsystems, effects of technological changes in subsystems and components on the overall system can be readily simulated and effects can be visualized.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and other features of the present invention will become readily apparent to those skilled in the art from the detailed description which follows and from the accompanying drawings, in which:

FIG. 1 illustrates an overview of the virtual system design environment of the computerized prototyping system of the present invention;

FIG. 2 illustrates components of the virtual system design environment;

FIG. 3 shows a hierarchical representation of a conceptual design process;

FIG. 4 shows components of an optimization and decision support system module;

FIG. 5 is a functional diagram of a design evaluation process;

FIG. 6 illustrates a typical decision tree in the optimization and decision support system module;

FIG. 7 shows fuzzy membership functions of variables;

FIG. 8 illustrates the variable interdependency in the analytical hierarchical process;

FIG. 9 shows a judgement matrix of the analytical hierarchical process;

FIGS. 10(*a*) and 10(*b*) illustrate power and acceleration utility functions, respectively;

FIG. 11 shows an example of an optimization and decision support system module front end graphical user interface;

FIG. 12 illustrates an optimization and decision support system module user interface for building decision trees, variable strength and performance objectives;

FIG. 13 shows an example of a decision tree in the optimization and decision support system module and the evaluation results of six alternative designs;

FIG. 14. is a functional flowchart for the optimization module;

FIG. 15 shows the architecture of a feedforward neural network;

FIG. 16 shows the incorporation of nonlinear functions in a decision model;

FIG. 17 shows an example of a graphical result of an optimization process;

FIG. 18 is a functional flowchart of the virtual system design environment detailed design stage;

FIG. 19 is a sample system hierarchical graph;

FIG. 20 is a sample subsystem information database of the system hierarchical graph;

FIG. 21 illustrate a robot arm having two links and three joints;

FIG. 22 illustrates a system hierarchical graph of the robot arm shown in FIG. 21;

FIG. 23 shows the subsystem information database corresponding to the system hierarchy graph shown in FIG. 22;

FIGS. 24(*a*) and 24(*b*) are representations of the hierarchical relationship in the subsystems and components, FIG. 24(*a*) being the two design and analysis path supported by the virtual system design environment, and FIG. 24(*b*) being an internal representation of the system subsystem designs;

FIG. 25 is an illustration shown the example of an interface of the geometric design browser;

FIG. 26 shows the selection of one of the subassemblies and corresponding geometric models;

FIG. 27 is an example interface of a functional design browser;

FIG. 28 shows the selection of one of the subassemblies and corresponding functional models;

FIG. 29 illustrates design modification under the virtual system design environment;

FIG. 30 is a functional flowchart of the virtual system design environment detailed system analysis stage;

FIG. 31 illustrates the interaction between the detailed system design stage and conceptual design stage during the detailed system analysis operation;

FIG. 32 is a model dependency matrix for the example shown in FIG. 30;

FIGS. 33(*a*) and 33(*b*) show submatrices for the model dependency matrix;

FIG. 34 shows an example interface of the system analysis stage; and

FIG. 35 is a schematic representing the general interrelationships between the human user and the computerized prototyping system of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Initially, attention is directed to FIG. 35 which shows a schematic representation of the general interrelationships between the human user and the computerized prototyping system of the invention, including the various components and associated interfaces. Stored in a computer RAM of the prototyping system are graphic software code, design browsers code, constraint solver program code, printer and plotter drivers, optimization algorithm code and knowledge base system code. Further, a computer hard drive contains the requisite database. Input devices include a keyboard and mouse, and a printer and plotter may be plotter may be provided in addition to the user interface screen.

As shown in FIG. 1, the virtual system design environment of the present computerized prototyping system includes four major stages of operations: (a) Downselection of candidate designs; (b) Conceptual level design optimization; (c) Virtual design and prototyping; and (d) Virtual analysis of the designed systems (FIG. 1). The candidate design downselection and higher level design optimization stages are a part of the conceptual design process, while the virtual prototyping and virtual analysis stages correspond to the detailed design stages. In the conceptual design stages, exact mathematical representation of the model to be optimized need not be known. The user can build a model using the decision tree principles, with the interrelationships between the variables defined as fuzzy variables, such as mediums, high, very high etc. The conceptual design stage of VSDE is handled by a module called Optimization and Decision Support System (ODSS). ODSS allows the downselection of alternate designs and their high level design optimization.

The detailed design for the subsystems is done in the virtual prototyping stage, where solutions are estimated for a set of equations describing the subsystems. As shown in FIG. 2, in VSDE, these operations are a part of the Design Environment (DE) module. The designs always need not be performed inside the VSDE, the can be designed outside and ported into from commercial CAD systems. In the case of functional design tasks, the subsystem functionalities can be incorporated into VSDE either through mathematical representations or through models. If performance of design at any stage in VSDE does not meet the designer requirements, the designer can easily go back to one of the previous design stages for modifying the decision model, the variable structure or the performance requirements. In the Virtual System Analysis stage, the overall integrity of the subsystems are checked and analyzed.

Stage 1: Evaluation of Design Alternatives

The ODSS is a decision making system for evaluation of candidate designs. In many engineering design applications, a set of alternative designs may be known initially, and the primary task will be to choose the most closely matching design with respect to a set of performance requirements and then modify it to satisfy the performance requirements. When the design process evolves from a very crude initial design, the evaluation process may not be needed at the beginning. But, when the detailed design results in more than one design, the user may be needing to evaluate the designs and re-optimize them. In VSDE, such loop back between design modules is easily incorporated.

In ODSS, the evaluation of alternatives and conceptual optimization of designs can be performed in stages. FIG. 3 shows such a staged conceptual design process. This allows the designer to build sub-dividing the conceptual design process. Each of these stages will be having a set of decision trees. Though variables in decision trees inside a stage can be interconnected, interconnection between variables in two different stages is not permitted.

Each stage will be associated with a set of specification database, based up on which the decisions are made. The underlying decision process in ODSS consists of a Knowledge Based System in conjunction with constraint satisfaction algorithms and evaluation/optimization methods. The system consists of four individual modules; a model base, a knowledge base, a constraint solver and a set of optimization routines. The individual modules will be interacting with each other and with the user through Graphical User Interface (GUI). The components of the overall system is shown in FIG. 4. The flow-chart representation of the evaluation methodology is shown in FIG. 5.

Knowledge Based System

The Knowledge Based System (KBS) consists of an Artificial Intelligence (AI) based representation of the decision hierarchy, dependencies of the variables at each of the hierarchies and associated strengths of each variable. The decision hierarchy is represented as a decision tree, and at any instant the characteristics of each of the nodes can be determined from the KBS and the input database.

The designer either builds a KBS using the ODSS or loads a previously designed KBS into the ODSS. During the KBS building, variables at each level and their interconnection strengths are provided through a text edit window. The designer views the decision tree structure through the graphical user interface and makes any necessary changes to the model, at any time during the design process.

The user has the flexibility to add or delete any variable at any level in the hierarchy, and determine its effect on the entire system. FIG. 6 details a typical decision tree in ODSS. In this case there are four levels of variable nodes. ODSS does not assume any restrictions on the number of levels or on the number of variables in each of the levels. For complex models and finer evaluations, the user expands the variables and associated structures, and perform the decision process.

In the decision tree shown in FIG. 6, there are four sublevels of variables and some of the sublevel variables are interconnected from other sublevel. Also, some of the sublevel hierarchies are deeper than the other ones. A deeper hierarchy corresponds to a finer decision model. Such flexibility in defining the interconnections and hierarchies allows ODSS to model complex decision and evaluation processes.

The interconnection strengths can be assigned in terms of numerical values or fuzzy variables. The numerical range for the strengths is between 0.0 and 1.0. The fuzzy strength corresponds to a set of membership functions defined in this numerical range. The designer can choose the number of variables defined in this range depending on the refinement needed. FIG. 7 shows a set of typical membership functions used in ODSS.

When numerical values are used for defining the variable strengths in the decision hierarchy, ODSS incorporates a scheme based on Analytical Hierarchy Process (AHP) (see Gass, S., *Decision making: models and algorithms*, Krieger Publishing, 1991.) for establishing a set of consistent connection strengths.

Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) provides a convenient way to build the decision model for downselection and optimization. The model is built by comparing the variables in a level pairwise. In FIG. 8, let V1, V2, V3 be three variables in a level, in one of the branches H1, in the decision tree.

The user assigns a set of pairwise weighting for the variables as shown below: Construct a 3×3 matrix with each element in the matrix representing the relative weights between the variable pairs. The weighting matrix will be n×n if the number of variables are n. For each row i of the matrix, take the product of the entries in that row (Π_i) and take the corresponding geometric mean P_i, where $P_i = 3\sqrt{\Pi_i}$
The normalized value of the geometric mean, P_i, denotes the priority or weight given to the i^{th} alternative (Vi), i.e.

$$p_i = \frac{P_i}{\sum_i P_i}$$

In the example of
FIG. 9, $P_1 = \sqrt[3]{a\overline{b}}$, $P_2 = \sqrt[3]{c/a}$, $P_3 = \sqrt[3]{1/bc}$. The consistency of the hierarchical model constructed by the user, can be checked by estimating a consistency index as follows.

Consider an AHP judgement matrix with n rows and n columns. Let p_i be the corresponding AHP priorities. Calculate the maximum eigen value of the matrix:

$$\lambda_{max} = p^1 \sum_{i=1}^{n} a_{i1} + p_2 \sum_{i=1}^{n} a_{i2} + \ldots + p_n \sum_{i=1}^{n} a_{in}$$

A consistency index CI can be represented by

$$CI = \frac{\lambda_{max} - n}{n - 1}.$$

This consistency index CI is then compared with a random consistency index RI, and a consistency ratio CR=CI/RI is calculated. RI is the average consistency index for a matrix of size n, whose reciprocal entries are drawn at random between 1/9 and 9. A weighing model is accepted if the consistency ratio CR is less than or equal to 0.10.

Constraint Solver

The constraint solver will be handling all the constraints to be satisfied in the decision process. The constraints can be of two types; required constraints or preferred constraints. The preferred constraints can have a hierarchy of strengths; for example, from very low to very high (see Zadeh, L.A., Fuzzy sets, *Information and Control*, 1965, pp. 338–353). A result which satisfies all the required constraints can be taken as an admissible one; which can be considered for a closer evaluation based upon the preferred set of constraints. In general, the constraint solver should be able to handle: adding a variable, removing a variable, adding a constraint and removing a constraint. The system should re-evaluate itself in these cases.

The property of the constraint solver to handle adding/removing of variables and constraints is very important in ODSS. The implications of future technology on subsystems as well as on the overall system will be incorporated through variables and constraints. Any constraint solving algorithm we develop or adopt should have this characteristics. Such an ability is also crucial in making the overall methodology very general to be used with design and downselection of very wide range of engineering systems.

Model Database

Each of the decision stages in ODSS has an associated model database. Model database consists of the specifications and functional details of the design alternatives under consideration. The database information provided needs to be consistent with the variable hierarchy defined in KBS. In ODSS, through the GUI, the designer can either build a new database or load an existing database. The interfacing is done through a text edit window, as in the case of variable hierarchy and strengths. The designer also provides the range of each of the database entries and the utility function to be used for converting the specifications into a common dimension.

Utility functions translate diverse input characteristics into a common scale. The range of utility curve encompasses the range of acceptable or realistic alternatives. The transformed range in ODSS is 0–1.0. The zero point for each utility curve indicates the level of performance which no longer provides effective value to the system performance. Each of the input variables in the decision model is associated with a utility curve.

In principle, there can be two types of input variables in a decision tree, a variable which produces positive effects for higher values, or a variable which produces negative effects for higher values. For example, in the design of a motor vehicle, power may be a positive variable while time for acceleration can be a negative variable. FIGS. 10(*a*) and 10(*b*) shows examples of utility functions for positive and negative variables. ODSS has a library of utility functions and the designer can specify one of them for each of the database variables.

The Graphical User Interface

The Graphical User Interface is an important component of ODSS and VSDE. In using the VSDE, many of the flexibility comes from user-friendly GUIS. Mainly there are three types of interfaces in ODSS. The front end where the designer selects all the necessary settings for starting the design and evaluation process; a set of text editor windows for building the decision model, variable hierarchy, database and specification of performance objective; a set of graphics panels to display intermediate as well as final results. The front end interface also allows loading and execution of predefined examples and decision models. FIG. 11 shows the front end interface of ODSS and FIG. 12 shows the text edit windows for defining variable hierarchy, strengths and performance objectives.

A sample set of decision trees in the first stage of alternative evaluation is shown in FIG. 13. In this case one of the second level variables is shared by two trees. The designer can scan through all the decision trees modeled in the ODSS and can make changes in the model through the GUIs for defining variable hierarchy. The sample results of an evaluation study is also shown in FIG. 13. In this case ODSS evaluated six different design alternatives against a set of designer specified criteria and ranked them in numerical order.

Stage 2: Conceptual Level Design Optimization

The second stage in VSDE consists of the optimization module for design-to-requirement tasks. If the designer has gone through the design evaluation process, he/she can pick one of the design alternative and optimize the design variables against a set of performance requirements. Otherwise, the designer can start with a crude set of design parameters and the decision model to obtain the optimized set of design parameters. As in the case of evaluation of alternatives, the optimization can be performed by splitting the whole process into different levels and optimizing each of them separately. While evaluation of prototypes is a forward chain operation, design optimization can be seen as a backward chain operation which is more goal-driven than data-driven.

The functional flow chart of the optimization process is shown in FIG. 14. The designer loads a decision model and specifies the performance requirements into the system, along with the specification database of the design to be optimized. As indicated before, the design can be one of the evaluated designs from the previous design stage or the database corresponding to a crude design. The specification database is iteratively modified using the optimization algorithms inside ODSS. ODSS has two optimization algorithms built in; one based on deterministic optimization principles and the other one based stochastic optimization principles.

One of the unique characteristics of the ODSS implementation is the translation of its knowledge base as a neural network architecture during the optimization process. This allows ODSS to use neural network optimization algorithms in the design process. Such an implementation also allows the ODSS to incorporate functional relationships, or models of subsystems into the high level system optimization process. Exact mathematical form of these models need not be known for the optimization process. These properties of ODSS and its use in subsequent system design stages is explained later on.

Another important characteristics of ODSS is the ability of its decision model to learn over design examples. Since the implementation of the KBS in ODSS is employed as a decision tree and algoritmically treated as a neural network,

learning from previous designs will not be difficult to incorporate. This feature is not currently employed in ODSS, but will definitely an attractive property to add on at later stages of development.

Decision Trees and Neural Networks

In ODSS, the architectural and operational similarities of neural networks and decision trees are exploited for the system optimization applications. FIG. 15 shows a feedforward neural network architecture with one layer of hidden neurons. In a typical feedforward neural network architecture, the nodes between two layers are fully interconnected. In the decision tree architectures used in ODSS, nodes in different sub-levels (FIG. 6) need not be fully interconnected; though in a complex decision model the number of interconnections may be high.

In some of the decision processes, all the decision trees need not be interconnected to all the other trees. In such cases, if the designer decided so, optimization of the individual decision trees and associated decision parameters can be done (when trees are interconnected, ODSS does not allow this option).

Deterministic Optimization Process

In ODSS an adapted form of bakpropagation algorithm (see Rumelhart, D., Hinton, G., and Williams, R., Learning internal representations by error propagation, in *Parallel Distributed Processing*, vol. 1., MIT Press, 1986) is used for the optimization process. Two different kinds of optimization are possible with the concepts set forth in ODSS. Either the optimization process can optimize the weights associated with each of the variables or it can optimize the value associated with each of the variables. In the first case, the decision model gets optimized and in the second case it is the input variables which are getting optimized. Only the second case is relevant for engineering design applications. But in applications where the decision model itself need to be optimized, the first characteristic of ODSS optimization will be useful. The designer can select this mode of optimization through the GUI.

Let net_i denote the summed output of a node i. Then,

$$net_i = \Sigma w_{ij} out_j$$

where w_{ij} is the interconnection weight from node j of the lower layer to the node i and out_j is the output of node j.

A sigmoidal transformation, f( ) to the summed output results in:

$$out_i = f(net_i) = \frac{1 - e^{-net_i}}{1 + e^{-net_i}}$$

When ODSS is used for decision model optimization, the updating equations for the interconnection strengths are expressed as:

$$W_{ij}(n+1)+W_{ij}(n)+\Delta W_{ij}(n+1)$$

where

$$\Delta W_{ij}(n+1)=\eta\delta_{pj}O_{pi}$$

When ODSS is used for design parameter optimization, the updating equations for the input variables are expressed as:

$$O_{pj}(n+1)=O_{pj}(n)+\Delta W_{pj}(n+1)$$

where,

$$\Delta O_{ij}(n+1)=\eta\delta_{pi}W_{ij}$$

where

$$\delta_{pi}=(t_{pi}-O_{pi})f(\text{net}_i)$$

for the output nodes and

$$\delta_{pi}=f(\text{net}_i)\Sigma\delta_{pk}W_{kj}$$

for any arbitrary hidden nodes

At each iteration, the parameters are updated as per the above set of equations.

One of the significant differences between the generally used neural network learning and the ODSS implementation is the ability of ODSS to incorporate any transformation function at the nodes. Nonlinear functions can be incorporated into the ODSS decision model by replacing the sigmoidal transformation function f(.) by the required one. For example, consider a node in the decision model represented in FIG. 16. In this case f(.) is a function of variables x_1, x_2, and x_3. x_1, x_2, and x_3 can be the input design parameters, in which case node N is in the input layer, or they can be the outputs of other nodes in the lower layer.

The second distinguishing property of the ODSS implementation of the optimization process compared to the Nonlinear Programming (NLP) approaches is that in ODSS, the exact functional form of f(.) need not be known for the overall optimization process. If the deterministic optimization process is used, the only requirement about function f(.) is that its derivative should exist and should be known. In cases where f(.) and its derivative are unknown, the designer can choose the stochastic optimization process in ODSS through the Graphical User Interface.

Stochastic Optimization Process

The second optimization algorithm implemented in ODSS is called Alopex (see Unnikrishnan, K. P., and Venugopal, K. P., Alopex: a correlation based learning algorithm for feedforward and recurrent neural networks, *Neural Computation*, vol. 6, 1994, pp. 469–490). Alopex is a simple, but powerful, stochastic optimization algorithm. The algorithm works by broadcasting a measure of the global performance, to all the nodes in the decision model, synchronously. The explicit derivative of the effort function need not be calculated for updating the parameters. A correlation measure between the change in weight and the global error change is estimated and the individual parameters are changed based on a probability index of going in the right direction of optimization, so that the global error function is minimized.

Consider a node i with an interconnection strength w_{ij}, from neuron j in the lower layer. The output of the node i, during the nth iteration, is given by:

$$\text{net}_i(n)=\Sigma W_{ij}(n)\text{out}_j(n)$$

Applying a transformation f(.) to this results in

$$\text{out}_i(n)=f(\text{net}_i(n))$$

During the nth iteration, output out_i is updated as,

$$\text{out}_i(n)=\text{out}_i(n-1)+\delta_i(n)$$

where δ_i(n) will have a small positive or negative step of size δwith the following probabilities:

$$\left.\begin{aligned}\delta_i(n) &= -\delta \text{ with probability } P_i(n)\\ &= +\delta \text{ with probability } (1-P_i(n))\end{aligned}\right\}$$

The probability P_i(n) is given by the expression:

$$P_i(n)=\frac{1}{1+\exp(-\Delta_i(n)/T)}$$

where Δ_i(n) is given by the correlation:

$$\Delta_i(n)=\Delta\text{out}_i(n)\Delta E(n)$$

Δout_i(n) and ΔE(n) are the changes in the weight $\text{out}_i$and the error measure E over the previous two iterations (for the first two iterations, P_i(n) is taken as 0.5). In the expression for P_i(n), T is a positive temperature that determines the effective randomness in the system. With a non-zero value for T, the algorithm takes biased random walks in the direction of decreasing E. The optimization process starts with a large value for the temperature T.

Stopping Criteria for the Optimization Process

In ODSS, the designer can specify the number of iterations (in GUI) through which the optimization process need to be continued or the error tolerance allowed in performance objectives. ODSS monitors the effort between the desired performance requirements specified by the designer and the system performance, and if this is less than the tolerance specified, the optimization process is stopped. Error tolerance based stopping overrides iteration based stopping. Iteration based stopping is necessary, since in some cases the designer may be putting unattainable performance specifications and there may not be a system designable for those specifications. In these cases, ODSS stops the optimization process based on the number of iterations specified by the designer.

Graphical User Interfaces Corresponding to the Optimization Process

The designer lists the performance specifications in the objectives text edit window on the front end GUI. If the optimization process is preceded by the evaluation of the alternatives, the designer can choose one of the alternative designs for optimization. The selection of stopping criteria and the choosing of the optimization algorithm (deterministic or stochastic) are also done through the front end GUI. The designer can observe the optimization process through windows and visualize the optimized system characteristics through the results window. FIG. 17 shows an example of the results display.

Sensitivity Analysis

The sensitivity of each of the variables in the decision model to the changes in system specifications can be easily calculated and displayed in ODSS. The designer can choose one of the input variables and find its effect on the overall performance requirements or any intermediate level decision model nodes, fixing all the other input variable values. The results are displayed on a graphics window in ODSS.

Stage 3: Detailed Design

The third stage in VSDE consists of the Detailed Design stage. This is a part of the Design Environment (DE). In this stage, subsystems and components of the system are designed and integrated into VSDE. FIG. 18 shows the functional flow graph of the detailed design stage. The designer has the flexibility to design some of the components inside VSDE itself or it can import externally designed components into it for integration. For internal design, the exact mathematical equations representing the subsystem being designed need to be known.

In VSDE, two different design and integration are possible; geometry based design and function based design. The modes of design can be chosen by the designer depending on the application in hand. In the geometry based design, the subsystem property of interest is the geometrical characteristics. The subsystems may be represented as 2D or 3D CAD models and the design environment allows the designer to virtually integrate the components into subsystems, and the subsystems into the overall system.

In the function based design, the property of importance is the functionalities of the subsystem and its components. This mode of operation in VSDE allows the designer to easily integrate subsystems such as electrical power supply, control systems, suspension etc. into the design environment. In this case the integration is based on the function models or mathematical equations representing the subsystems or its components.

In both the modes of operation, each of the subsystems are associated with a Subsystem Hierarchy Graph (SHG) and a Subsystem Information Database (SIDB). The Subsystem Hierarchy Graph dictates the hierarchy of subsystems and any changes in subsystems, components and their integration need to be reflected through SHG. SIDB keeps a record of the inter-relationships with other subsystems and associated constraints. At any time during the design process, the designer can change the information database, subsystem models or functionalities and do the studies based on the new set of subsystems and functionalities. Also, the designer can study the effect of changes in each of the subsystems on the overall performance characteristics he/she has specified in ODSS during the Stage 1 or Stage 2 operation.

System Hierarchy Graph

During the detailed design stage, the subsystems and components of the subsystems are represented by the SHG. Each of the subsystems and its components are represented in a hierarchy with each link representing the breakdown of the corresponding entity. Each rectangle in SHG corresponds to an entity and each of the rectangles connected to it from the lower level corresponds to the primitive entities associated with it. For example, an entity car can have engine, chassis, wheels etc. as the primitives and at the next lower level engine may be having primitives such as casing, spark plugs, fuel injection module, and valves. FIG. 19 shows a simple SHG of a power distribution system represented as a three level hierarchy. The user can define more levels in the hierarchy primitives for each of the entities, depending on the refinement in the model needed.

Subsystem Information Database

The Subsystem Information Database (SIDB) consists of a connection matrix relating the subsystems and components themselves. For example, let Power Distribution Circuits (PDC) module in FIG. 21 have a direct correspondence with the module Battery. The correspondence means, for effective functioning of PDC, Battery is a must and vice versa. Note that in the SHG this information is not reflected as interrelationships between modules in the same level. SIDB helps in simulating the overall system by collectively considering the subsystems and their interrelationships. When VSDE is used for evaluating technology changes in subsystems and components, such a database is essential. Technological advances in one of the subsystems need not translate to an overall improvement on the systems, unless advances of same magnitude have happened with the other related subsystems and components.

For the example hierarchy in FIG. 19, the SIDB for the middle level will be of the form shown in FIG. 20. Here, the entry (Battery, Wheel Drive) relates the importance of Battery for the function of Wheel Drive, and the entry (Wheel Drive Power Distribution Circuit) relates the importance of Wheel Drive for the function of Power Distribution Circuit. Note that the matrix need not be symmetrical, since the relationships between subsystems or relationships between components need not be symmetrical. For example, (Battery, Wheel Drive) relationship may be High, while (Wheel Drive, Battery) relationship need not be High.

The SIDB in the case of geometrical design and functional design will be different. In the case of geometrical design, SIDB contains information about the physical linking of subsystems and components. Hence the SIDB matrix in this case will not be having fuzzy entries such as Medium, Low etc. The relationships will be expressed as yes or no, with the yes value indicated by a 1 and a no value indicated by a 0. Also in geometrical design, the SIDB matrix will be symmetrical.

An example of SIDB in the case of geometry based design is given below. FIG. 21 shows a robot arm with two links, three joints and a set of fingers. The SHG for this system will be of the form as shown in FIG. 22. SIDB plays an important role in applications like this. In this case SHG does not convey information about the arrangement of the links; whether link 1 and link 2 are connected through joint 1 or joint 2; or whether the wrist is connected to joint 3 or joint 2 etc. A typical SIDB for this design problem is shown in FIG. 23. The matrix elements are 1, if the corresponding (row, column) combination is physically connected and 0, if not.

In both these examples, the SIDB is represented as a matrix with the relationships between two entities represented as a single variable or numerical value. In the present implementation, this is done employing an array. More sophisticated implementation of SIDB may be needed to deal with more complex problems. In such cases each of the relationships may be represented by a multiple element data structure.

Geometry Based Design and Functional Design

VSDE allows two types of system design and analysis, geometry based design and functional design. In geometry based design, the physical characteristics of the system is taken into consideration. It is similar to a typical top-down design approach. The physical characteristics include size and shape. In this case, the subsystems are the physically separable components of the system. The system can be represented as a 3D structure or 2D structure and VSDE allows incorporation externally designed components of these structures. The objective in this case is to evaluate the integration of subsystems and their components and visualize them graphically. The subsystems are of the form of CAD models.

In functional design, the functional integrity of the system and its subsystems are simulated and evaluated. VSDE supports mathematical representation or input/output mapping of the subsystem functionality. The input/output mapping can be of the form of a black box inside which the mapping is represented as a neural network, fuzzy logic system or knowledge based representation. All the subsystems and components need not be modeled using the same modeling technique. Some of the subsystems can be represented mathematically, some of them represented as neural network models or some others as knowledge based systems. This characteristics of VSDE is very useful in system integration studies because, in many cases subsystems may not be designed by a single agency and some of the subsystems may be off-the-shelf products.

In both geometry based design as well as functional design, SHG and SIDB defines the system structure and interdependencies (FIG. 24). The geometric details as well as functional details of a design are two parallel aspects of SHG and SIDB (FIG. 26*a*). At any time the designer can view and edit the geometric details or functional details using corresponding design browsers. The design browsers are called Geometry Design Browsers (GDB) and Function Design Browsers (FDB). For making any changes in these browsers, the designer need to make corresponding changes in SHG and SIDB.

Geometry Design Browsers

The Geometry Design Browser (GDB) is an interactive editor that provides access to each individual components of a system. The designer can edit the SHG and SIDB also in

GDB. FIG. 25 shows an example user interface for GDB. It has three display windows and two panel windows. The designer can choose one of the three display windows for editing. For example, he/she can choose the Model Window and select any of the subsystems or components and arrange them anyway he/she decides. Corresponding changes are made on the SHG and SIDB windows so that the overall system is consistent with the hierarchy. Each of the subsystems have an associated CAD model and the designer can select any one of these subsystems and edit its hierarchy and components. For example, FIG. 26 shows the interface when the designer has selected the Head Assembly subsystem. In this interface only the selected subsystem and its components will be displayed and can be edited.

Thus the designer can scan through each of the levels in the hierarchy or each of the subsystems in each of the hierarchy levels and do detailed design studies. If any of the geometric details of the subsystems are not acceptable, the designer will be able to redesign that subsystem and virtually reintegrate the whole system again.

The GDB can incorporate 2D or 3D models of the subsystems. In the case of 3D models, the designer can select any perspective view of the design by specifying the look angle.

Implementation

The GDB is implemented in an Object Oriented environment. Each of the subsystems and their components are treated as an object. Depending on the hierarchy specified in SHG and relationships specified in SIDB, some of these objects will be inheriting characteristics from their parent objects. The model rendering and associated graphics manipulations are done through World Toolkit Libraries.

Function Design Browsers

Like GDB, Function Design Browser (FDB) is an interactive editor which allows the designer to design and integrate system functionalities. The underlying subsystem relationships are represented by the SHG and the SIDB. FIG. 27 shows an example FDB for the system shown in FIG. 19. As in the case of GDB, FDB also has three display windows and two panel widows. The first difference between GDB and FDB is in the model window. In the case of functional design, in this window the system functionalities are represented with all the interrelationships. The second difference is that dynamic simulation of the functionalities is possible in FDB. Such a feature is not currently incorporated in GDB.

Each of the subsystems are represented mathematically or through input/output models. VSDE supports three different kinds of models; neural network representation, fuzzy logic systems and knowledge based system models. The design environment is flexible enough to support a combination of all the three models and mathematical representation together. This is a unique property of VSDE.

Since the subsystems need not be designed inside VSDE, the designer may not be having control over the representative format of some of the subsystems. Some of the subsystems may be off-the-shelf products. The design environment supports integration of such hybrid subsystems and their evaluation. Also, this property of VSDE is essential when it is used as a technology evaluation tool. The technological advancements in subsystems may be difficult to be predicted and the system evaluation tool should be able to integrate and study subsystems from different domains.

Some of the subsystems which has only geometrical properties associated with them, will not be showing up on the functional design browsers. For example, in Power Distribution Circuits subassembly, the components Casing and PCB may be having only geometrical properties associated with it (unless, Casing is used for heat dissipation, grounding etc., or electrical resistance of PCB circuits is relevant; in which case they have electrical functions associated with it). If the designer chose to examine this subsystem, the FDB will look like the one in FIG. 28.

Unlike GDB, implementation of FDB does not require sophisticated graphics manipulation libraries or display routines. The important software components of FDB are the Input Data Interface, Sub-System Model, and Output Data Interface. The designer has to specify the type of modeling employed (mathematical representation, neural networks etc.) so that the browser can make necessary changes in the data interfacing. During the simulation stage, the designer can monitor the input data, output data and any data flow at any intermediate point of interest, specified through the user interface.

FDB allows two modes of operation; an off-line mode and an on-line mode. In off-line mode, the input data is provided from a file and the output data as well as the specified intermediate results, are stored as data files. These data files can be visualized as graphs or charts later on. In the case of on-line mode the designer can visualize the results at the output and intermediate points through on-line visualization windows.

Implementation of Geometric and Function Design Browsers

The Design Browsers are implemented in an Object Oriented environment with C++/Java as the programming environment. The user interfaces are developed in X/Visual C++.

Stage 4: Detailed System Analysis

The final stage in VSDE is the analysis of the overall system. In Stage 3, VSDE allows the designer to design and integrate the subsystems and the components of subsystems. Stage 3 does not allow the designer to analyze the whole system with respect to the objectives and constraints set forth in Stages 1 and 2. This feature in integrated in Stage 4 of VSDE. Since all the design stages are integrated, the designer can go back to any of the earlier design stages and make changes to the subsystems, components, performance characteristics or design goals and re-evaluate the entire system. This feature of VSDE is extremely useful in applications involving design process consisting of many iterative steps. Also, the entire system or any of its subsystems can be simulated structurally as well as functionally during this design process and the design parameters can be optimized. The representation of such a stages approach in VSDE is illustrated in FIG. 29. Note that the Stages 1 and 2 can have multiple substages in each of them (FIG. 3) and VSDE allows the designer to go back to any of these substages during the Stage 4 analysis.

The functional flow chart of Stage 4 is shown in FIG. 30. As in the case of the previous stage, analysis can be done based on geometry or functionalities. The characteristics of the system, subsystem and their components form the inputs to the ODSS and for the designed system the designer can check whether the top level objectives are met. The number of system characteristics needed for the evaluation depends on the ODSS model complexity. In some cases, the ODSS model may be incorporating inputs from the subsystem levels too. Also, all the inputs to the ODSS need not be the results of detailed design. Some of the inputs to ODSS may be from external sources.

The above aspect of VSDE is explained in FIG. 31. Consider the example of Power Distribution System design (FIG. 21). Let Cost of the system is the higher level variable of most concern. During Stage 1 and 2 of the design process, the designer would have made a decision model (as decision trees) representing the higher level variables (FIGS. 6 & 13), including Cost. If Cost is the only variable of concern, there will be only one top most level node in the ODSS model. The inputs to the model will be design parameters such as size of battery, weight of battery, motor power, motor size etc., which are the parameters decided during the detailed design phase in Stage 3. The ODSS model can also have external inputs such as cost of assembly, shipment cost/lb. etc. which are not design parameters.

During the functional or geometric design stage, each of the nodes in the functional hierarchy or geometric hierarchy will be associated with more than one design parameters. For example, the node Motor may be associated with size, weight and power; the node Casing may be associated with size and weight. The number of design parameters associated with a node depends up on the refinement of the functional or geometric models. The node Motor by itself can represent a model hierarchy consisting of subnodes such as weight, size and power. It is up to the designer to decide how refined the model should be and the design environment imposes no restriction on the complexity of the models being built or the design tasks being considered. As indicated before, such a hierarchical representation of subsystems and components, and their implementation based on Object Oriented principles provide VSDE considerable flexibility in modeling and power on the overall design process.

Model Dependency Matrix

The interconnections of design parameters-to the ODSS models are defined through a Model Dependency Matrix (MDM). The MDM is similar to the SIDB in Stage 3, except that each of the elements in MDM may be associated with a submatrix. This submatrix defines the interconnections of subnodes associated with that node. Such submatrices will exist only when multiple attributes are associated with the nodes in design hierarchy. For example, FIG. 32 shows the MDM for the example shown in FIG. 30. The elements (Motor, var 8), (Battery, var 6), (Battery, var 2), (PCB, var 6) and (Casing, var 5) have submatrices associated with them. FIG. 33 shows some of those submatrices.

Implementation

Following the detailed design, the designer can analyze the designed system against each of the conceptual level objectives (specified in ODSS, during Stage 1 and 2). The designed system need not be the same as the ODSS optimized system because of geometrical or functional limitations of the subsystems and components. Some of these limitations may be surfacing only during the detailed design phase.

The VSDE user interface for Stage 4 allows the designer to go back to the ODSS knowledge based model and decision trees and change the hierarchy or variables, or change the operations in detailed design stage (Stage 3). FIG. 34 shows an example user interface for Stage 4. As in Stage 3, this interface also has three display windows and two panel windows. The display windows show the SHG/Function Mode/Geometric Model, the decision model used in ODSS and the Model Dependency Matrix. Clicking on the SHG/Model window allows the designer to go back to Stage 3, for the detailed design process. Clicking on the ODSS decision model allows the designer to change the decision model, and clicking on the MDM window allows changing of the MDM matrix. Consistency of the changes is checked each time a change has happened and the information contained in all the three display windows need to be consistent with each other.

What is claimed:

1. A computer-implemented prototyping method comprising the steps of:

identifying and selecting a design from a collection of alternate designs which best satisfies a set of conceptual level design specifications;

optimizing characteristics of the design based on the conceptual level design specifications;

modifying the specifications interactively using graphical interfaces for reevaluating and re-optimizing the designs;

simulating the functional and geometrical properties of the design and components of the design on a computer using graphical design browsers;

analyzing a performance of the design against a set of design specifications and effecting a redesign by interactively selecting one of the previous design operations through the graphical user interfaces;

wherein said identifying step includes:

using a knowledge base system including a user modifiable artificial intelligence based representation of a decision tree defined by an interconnected multi-level nodal hierarchy having a strength assigned to each of the interconnections between nodes in the hierarchy;

using a database including a stored database representation of the functional details of the design alternatives under consideration by the user, the database representation being consistent with the multi-level nodal hierarchy of the knowledge based system;

using a constraint solver implemented as a set of logic programs which evaluates the user specified constraints on the nodal variables in the nodal hierarchy against the functional details of the design alternatives represented in the database and computed nodal strengths during an evaluation process;

using a graphical user interface including a first edit window for designing, displaying and revising the decision tree in response to user input commands through a computer keyboard and mouse; and a second edit window for displaying the database representation and revising the database representation in response to user input commands; and a third edit window for displaying and revising the specifications against which the design are evaluated in response to user input commands; and further including a set of graphics panels for displaying intermediate and final results of the evaluation process;

wherein the said optimization step includes:

using a neural network based representation of the nodal hierarchy with the interconnection strengths of the hierarchy implemented as weights in the neural networks and the user specifications represented as an objective function at the output layer of the neural network, and the input variables corresponding to the functional details of the design being optimized stored in the database;

using another graphical user interface including a front end for the user to specify the number of cycles through which the optimization is performed, and yet another graphical interface for specifying whether the optimization is performed on a single design or on a set of designs, and a set of colored graphics panels for displaying the intermediate and final results of the optimization, and another graphical interface to allow the user to revise design specifications and re-optimize the designs.

2. A method according to claim 1, wherein said simulation step includes:

solving a set of mathematical equations representing the functional behavior of the components, implemented using software programs residing in the computer memory, for design of components of the design;

incorporating components designed externally, without the help of software programs mentioned above, into the computer memory;

specifying the interrelationships of design components in terms of their functional properties as well as geometrical properties employing a set of matrix based databases and hierarchy graphs defined by a multi-level nodal hierarchy having each of the nodes representing a component of the design;

browsing the functionality and geometry of the design using a set of graphical interfaces in which the designs, its functional hierarchy, and its geometrical hierarchy are displayed; in which the user can select any one of the components shown on the display and examine or revise corresponding hierarchy.

3. A computer-implemented prototyping method comprising the steps of:

identifying and selecting a design from a collection of alternate designs which best satisfies a set of conceptual level design specifications;

optimizing characteristics of the design based on the conceptual level design specifications;

modifying the specifications interactively using graphical interfaces for reevaluating and re-optimizing the designs;

simulating the functional and geometrical properties of the design and components of the design on a computer using graphical design browsers;

analyzing a performance of the design against a set of design specifications and effecting a redesign by interactively selecting one of the previous design operations through the graphical user interfaces;

wherein said simulation step includes:

solving a set of mathematical equations representing the functional behavior of the components, implemented using software programs residing in the computer memory, for design of components of the design;

incorporating components designed externally, without the help of said software programs, into the computer memory;

specifying the interrelationships of design components in terms of their functional properties as well as geometrical properties employing a set of matrix based databases and hierarchy graphs defined by a multi-level nodal hierarchy having each of the nodes representing a component of the design;

browsing the functionality and geometry of the design using a set of graphical interfaces in which the designs, its functional hierarchy, and its geometrical hierarchy are displayed; in which the user can select any one of the components shown on the display and examine or revise a corresponding hierarchy.

4. A computer-implemented prototyping method comprising the steps of:

identifying and selecting a design from a collection of alternate designs which best satisfies a set of conceptual level design specifications;

optimizing characteristics of the design based on the conceptual level design specifications;

modifying the specifications interactively using graphical interfaces for reevaluating and re-optimizing the designs;

simulating the functional and geometrical properties of the design and components of the design on a computer using graphical design browsers;

analyzing a performance of the design against a set of design specifications and effecting a redesign by interactively selecting one of the previous design operations through the graphical user interfaces;

wherein said identifying step includes:

using a knowledge base system including a user modifiable artificial intelligence based representation of a decision tree defined by an interconnected multi-level nodal hierarchy having a strength assigned to each of the interconnections between nodes in the hierarchy;

using a database including a stored database representation of the functional details of the design alternatives under consideration by the user, the database representation being consistent with the multi-level nodal hierarchy of the knowledge based system;

using a constraint solver implemented as a set of logic programs which evaluates the user specified constraints on the nodal variables in the nodal hierarchy against the functional details of the design alternatives represented in the database and computed nodal strengths during an evaluation process;

using a graphical user interface including a first edit window for designing, displaying and revising the decision tree in response to user input commands through a computer keyboard and mouse; and a second edit window for displaying the database representation and revising the database representation in response to user input commands; and a third edit window for displaying and revising the specifications against which the design are evaluated in response to user input commands; and further including a set of graphics panels for displaying intermediate and final results of the evaluation process;

wherein said simulation step includes:

solving a set of mathematical equations representing the functional behavior of the components, implemented using software programs residing in the computer memory, for design of components of the design;

incorporating components designed externally, without the help of software programs mentioned above, into the computer memory;

specifying the interrelationships of design components in terms of their functional properties as well as geometrical properties employing a set of matrix based databases and hierarchy graphs defined by a multi-level nodal hierarchy having each of the nodes representing a component of the design;

browsing the functionality and geometry of the design using a set of graphical interfaces in which the designs, its functional hierarchy, and its geometrical hierarchy are displayed; in which the user can select any one of the components shown on the display and examine or revise corresponding hierarchy.

5. A method according to claim 4, wherein the said analyzing step includes:

integrating the hierarchical representation of the properties with the knowledge base system and its decision tree hierarchy, implemented in the computer memory;

analyzing the sensitivity of each of the components against the design specifications represented through the knowledge base system, its nodal hierarchy, and against the user specified constraints;

visualizing the performance of the design graphically on the computer screen using multiple graphics panels, which capability for the user to interactively select each of the components of the design and visualize its functional as well a geometrical performance;

the user to stop the analysis process and store the conditions on to the computer memory, and graphically select any of said step, and revise associated specifications, nodal hierarchy, components hierarchy or the databases; and independently perform each of the steps with the conditions stored in the computer memory.

6. A computerized prototyping system comprising:

identifying means for identifying and selecting a design from a collection of alternate designs which best satisfies a set of conceptual level design specifications;

optimizing means for optimizing characteristics of the design based on the conceptual level design specifications;

modifying means for modifying the specifications interactively using graphical interfaces for re-evaluating and re-optimizing the designs;

simulating means for simulating functional and geometrical properties of the design and components of the design on a computer using graphical design browsers;

analyzing means for analyzing a performance of the design against a set of design specifications and effecting a redesign by interactively selecting one of the previous design operations through the graphical user interfaces;

wherein said identifying means includes:

a knowledge base system including a user modifiable artificial intelligence based representation of a decision tree defined by an interconnected multi-level nodal hierarchy having a strength assigned to each of the interconnections between nodes in the hierarchy;

a database including a stored database representation of the functional details of design alternatives under consideration by the user, the database representation being consistent with the multi-level nodal hierarchy of the knowledge based system;

a constraint solver implemented as a set of logic programs which evaluates user specified constraints on nodal variables in the nodal hierarchy against the functional details of the design alternatives represented in the database and computed nodal strengths during an evaluation process; and

a graphical user interface including a first edit window for designing, displaying and revising the decision tree in response to user input commands through a computer keyboard and mouse; and a second edit window for displaying the database representation and revising the database representation in response to the user input commands; and a third edit window for displaying and revising the specifications against which the design are evaluated in response to the user input commands; and further including a set of graphics panels for displaying the intermediate and final results of the evaluation;

wherein the said optimizing means includes:

a neural network based representation of the nodal hierarchy with the interconnection strengths of the hierarchy implemented as weights in the neural network and the user specifications represented as an objective function at the output layer of the neural network, and the input variables corresponding to the functional details of the design being optimized stored in the database;

another graphical user interface including a front end for the user to specify a number of cycles through which the optimization is performed, and yet another graphical interface for specifying whether the optimization is performed on a single design or on a set of designs, and a set of colored graphics panels for displaying the intermediate and final results of the optimization, and another graphical interface to allow the user to revise design specifications and re-optimize the designs.

7. A computerized prototyping system comprising:

a knowledge base system including a user modifiable artificial intelligence based representation of a decision tree defined by an interconnected multi-level nodal hierarchy having a strength assigned to each of the interconnections between nodes in the hierarchy;

a database including a stored database representation of the functional details of design alternatives under consideration by the user, the database representation being consistent with the multi-level nodal hierarchy of the knowledge based system;

a constraint solver implemented as a set of logic programs which evaluates user specified constraints on nodal variables in the nodal hierarchy against the functional details of the design alternatives represented in the database and computed nodal strengths during an evaluation process;

a graphical user interface including a first edit window for designing, displaying and revising the decision tree in response to user input commands through a computer keyboard and mouse; and a second edit window for displaying the database representation and revising the database representation in response to the user input commands; and a third edit window for displaying and revising the specifications against which the design are evaluated in response to the user input commands; further including a set of graphics panels for displaying the intermediate and final results of the evaluation;

a neural network based representation of the nodal hierarchy with the interconnection strengths of the hierarchy implemented as weights in the neural network and the user specifications represented as an objective function at the output layer of the neural network, and the input variables corresponding to the functional details of the design being optimized stored in the database;

another graphical user interface including a front end for the user to specify the number of cycles through which the optimization is performed, and yet another graphical interface for specifying whether the optimization is performed on a single design or on a set of designs, and a set of colored graphics panels for displaying the intermediate and final results of the optimization, and another graphical interface to allow the user to revise design specifications and re-optimize the designs.

* * * * *